Indian Institute of Technology Kanpur

SURGE 2023

Research Report

"EEG-Based Analysis of Cognitive States and Emotional Valence: Predictive Modelling, Brainwave Frequencies, and Dominant Brain Locations"

Submitted By:
Priyansh Singh
SURGE Roll no. 2330339
Department of Chemical Engineering
Indian Institute of Technology Kanpur
Kanpur, Uttar Pradesh

Under the guidance of:
Dr. Tushar Sandhan
Assistant Professor
Department of Electrical Engineering
Indian Institute of Technology, Kanpur
Kanpur, Uttar Pradesh

CERTIFICATE

This is to certify that the project entitled "**EEG-Based Analysis of Cognitive States and Emotional Valence: Predictive Modelling, Brainwave Frequencies, and Dominant Brain Locations**" submitted by **Priyansh Singh**(2330339) as a part of Summer Undergraduate Research and Graduate Excellence 2023 offered by the Indian Institute of Technology, Kanpur, is a bonafide record of the work done by him under my guidance and supervision at the Indian Institute of Technology, Kanpur from 11th May, 2023 to 12th July, 2023.

Dr. Tushar Sandhan
Assistant Professor
Department of Electrical Engineering
Indian Institute of Technology Kanpur, Kanpur

Acknowledgement

First and foremost, I am grateful to the institute IIT Kanpur, and my Project Supervisor and Mentor Prof. Tushar Sandhan for selecting me in the prestigious internship program SURGE 2023 and giving me the opportunity to work under his guidance.
I am deeply indebted to my Professor for emphasizing and encouraging me to work in the domain of EEG signals and brainwave frequencies which was very new to me and opened new doors of opportunity for me. I am thankful to him for guiding and mentoring me on projects, whenever I was stuck and for giving me a chance to explore my calibre.
I am also thankful to Ms Swathi Pratapa (Y22 MSR student) for helping me understand various pre-processing techniques used in signal processing and also guiding me through the usage of various filters in EEG-data processing.
I am extremely thankful to the Department of Electrical Engineering, IIT Kanpur for providing me with an environment that helped me develop an interest in research and further studies.
This internship would have been impossible without the constant support and inspiration of my family, who had faith in me, much more than I could have. They guided and motivated me throughout, helping me go through difficult days and inspiring me to work harder. Lastly, I would like to thank my sister, Ms. Swapnil singh (BTech Y19 electrical engineering) for her constant support.

Abstract

This report presents a comprehensive investigation into analysing cognitive states and emotion valence using electroencephalogram (EEG) data. The study encompasses four distinct tracks aimed at gaining insights into the prediction of cognitive states, exploring the relationship between EEG-based brainwave frequencies and different cognitive states, examining the association between brainwaves and their dominant locations in the brain, and predicting emotion valence states using EEG data. By leveraging EEG signals, this research aims to advance our understanding of cognitive processes and emotional experiences, contributing to various applications in cognitive science and effective computing.

Track 1: A public data set was utilised for this. It was made up of five subjects' combined 25-hour EEG records while they were performing a low-intensity control task. The task involved utilising the "Microsoft Train Simulator" program to steer a computer-simulated train. A Butter worth high pass filter and an STFT (Blackman window) pipeline were employed for data pre-processing. The model utilized features extracted from the EEG signals captured during focused, unfocused, and drowsy states. Finally, 4 models were compared –SVM-linear, SVM-RBF, KNN, and decision trees. The model achieved the best final accuracy of 94.31 % using K-nearest neighbours.

Track 2: This track analysed the relationship between EEG-based brainwave frequencies and their varying presence in different cognitive states. The same dataset of track 1 was used to cater for this. Through spectral analysis of EEG data, distinct patterns of brainwave frequencies associated with different cognitive states were identified, providing insights into attention, concentration, and alertness. The "Focused" state was marked with the dominant presence of the beta and alpha waves. The "Unfocused" state had decreased presence of the beta waves with increasingly dominant alpha and theta waves. "drowsy" state had a huge presence of the theta waves.

Track 3: In this track, the investigation explored the association between EEG-based brainwaves and their dominant locations in the brain. The dataset of track 1 was used for this purpose. The three brainwave frequencies: alpha, beta, and theta, were plotted using various electrodes corresponding areas of the brain. The presence of alpha waves was found to be significantly high in the parietal and occipital lobes(P7,P8,O1,O2) of the brain. Similarly, the presence of the low-beta waves was found significantly high in the frontal lobes of brain(F7,F3,AF4).

Track 4: A different EEG dataset was utilized for emotional valence prediction. The data was collected from two people (1 male, 1 female) for 3 minutes per state -positive, neutral, and negative. Stimuli to evoke emotions were given in the form of visual movie clips marked as positive and negative. This was used to predict emotion valence as positive, negative, or neutral based on the EEG patterns. The results demonstrated the feasibility of using EEG data for emotion valence prediction, providing valuable insights into the neural correlates of emotions.

Contents

List of Figures

1 Introduction

Neuroscience has proved that continuous exposure to mental load may affect brain dynamics. These variations are reflected in the electroencephalography (EEG) signal. Therefore, analysis of EEG signals is often utilized for cognitive load recognition. It is a non-invasive method for cognitive load detection. EEG represents the electrical activity of the brain that varies if there is any change occurs in the input conditions. Other physiological markers can only record secondary variations created by the brain, which is the fundamental response source to any environmental input. In Track 1, such variations in EEG signal under the exposure of a low-intensity control task are detected and analyzed in order to classify the overall brain response into three relevant classes i.e. focused, unfocused and drowsy state.

Detecting mental state changes in human individuals who need to remain dormant or passive while also having to maintain a continuous significant level of concentration or attention is the problem of the day. An example of such a scenario can be supervising automated processes or systems. Another example can be controlling robotic vehicles or drones or security monitoring. Yet another example can be the long-term monitoring of aircraft pilots while under the control of the autopilot. In all these cases, non-interfering supervision of processes is desired, while also alertness and a quick reaction is required of the involved individuals. With this study, we aimed to demonstrate that it is possible to identify pure mental states such as engaged and focused attention versus detached and unfocused monitoring from EEG data and tried to develop a machine learning-based system for solving such a task. Previous studies have used extra data (e.g., task engagement index or respiration data) to support EEG signals or limited the number of mental states detected to two to achieve high accuracy. We tried to detect the differentiation of engaged/focused, detached/unfocused, and drowsing mental states experimentally in a version of continuous performance tasks with 94.30 per cent (best) accuracy by using only the EEG data.

Track 2 focused on analyzing the relationship between EEG-based brainwave frequencies and their varying presence in different cognitive states . Brainwaves are divided into delta (0–4 Hz), theta (4–8 Hz), alpha (8–12 Hz), low beta (12–16 Hz), high beta (16–25 Hz), and gamma (25–50 Hz) waves, depending on their frequency. The intensity of brainwaves varies according to the state of human mental activity, and the degree of human cognitive load can be estimated based on this measurement. Theta waves are present in deeply relaxed, inward focused states with delta waves being prevalent in a sleep state. Alpha waves appear mainly in relaxed with passive attention, beta waves appear in problem solving, and gamma waves appear mainly in more complex mental functions.

The presence of alpha waves is found to be significantly high in the parietal and occipital lobes of the brain. Thus, the EEG electrodes in that locations show a dominant presence

of alpha waves. Similarly, presence of the low-beta waves is significant in the frontal lobes of the brain. These facts were verified in our Track3.
Emotion is a complicated condition that expresses human awareness and is described as a reaction to environmental stimuli. Emotions are, in general, reactions to ideas, memories, or events that occur in our environment. Russell's 2D emotion model is used most frequently. The vertical axis represents the arousal dimension (expressing the emotional intensity of the experience, ranging from low to excitement), and the horizontal axis represents the valence dimension (showing the degree of cheerfulness or joy, ranging from negative to positive). There are four categories of emotions in the arousal-valence coordinate system. The negative emotions are represented on the left side of the coordinate and the positive emotions are shown on the right. The valence axis represents both positive and negative emotions, and the arousal axis varies from inactive to active emotions(intensities). Thus, our Task 4 focused on emotional valence states prediction as positive, negative and neutral.

2 Dataset

2.1 Track-1,2,3

It was made up of five subjects' combined 25-hour EEG records while they were performing a low-intensity control task. The task involved utilising the "Microsoft Train Simulator" program to steer a computer-simulated train. In each trial, participants used the aforementioned computer simulation programme to operate a train for 35 to 55 minutes over a largely featureless course. Specifically, during the first 10 min of each experiment, the participants were engaged in focused control of the simulated train, paying close attention to the simulator's controls, and following the developments on the screen in detail. During the second 10 min of the experiments, the participants stopped following the simulator and became de-focused. The participants did not provide any control inputs during that time and stopped paying attention to the developments on the computer screen; however, they were not allowed to close their eyes or drowse. Finally, during the third 10 min of the experiments, the participants were allowed to relax freely, close their eyes and doze off, as desired. The three mental states that were looked at in this study were passive but focused attention (0–10 minutes of the job), alert but unfocused (10–20 minutes of task), and drowsy (20– 30 minutes of task). For each participant, the experiment was conducted for 7 continuous days(35-55 minutes recording on each day). The data set consisted of 14 EEG-electrodes data.

2.2 Track-4

The data was collected from two people (1 male, 1 female) for 3 minutes per state - positive, neutral, and negative. We used a Muse EEG headband which recorded the TP9, AF7, AF8 and TP10 EEG placements via dry electrodes. Six minutes of resting neutral data is also recorded, the stimuli used to evoke the emotions are below. 1-minute clips were shown of all the movies to both the users.

Table 1: Stimulus Valence Studio Year

Stimulus	**Valence**	**Studio**	**Year**
Marley and Me	Neg	Twentieth Century Fox, etc.	2008
Up	Neg	Walt Disney Pictures, etc.	2009
My Girl	Neg	Imagine Entertainment, etc.	1991
La La Land	Pos	Summit Entertainment, etc.	2016
Slow Life	Pos	BioQuest Studios	2014
Funny Dogs	Pos	MashupZone	2015

3 Pre-processing tools

3.1 Butter-worth filter

- A Butterworth highpass filter belongs to the class of infinite impulse response (IIR) filters, which means it has **feedback** in its design. It is characterized by its maximally flat frequency response in the pass band, meaning it **introduces minimal distortion** to the signal within the desired frequency range.

- It is designed based on the Butterworth filter's transfer function, which is determined by the order of the filter and its cutoff frequency. In our approach we have used a **5th order filter** with a cut-off frequency of **0.16Hz**. The order of the filter defines the **steepness of the roll-off**, or how quickly the filter attenuates frequencies below the cutoff frequency. A higher order filter will have a steeper roll-off but may introduce **more phase distortion**.

- The cutoff frequency of the highpass filter determines the frequency below which the signal is attenuated. Frequencies above the cutoff frequency are passed with minimal attenuation. The Butterworth highpass filter achieves a maximally flat frequency response in the **passband**, meaning there is no peaking or ripple in the passband.

- One important characteristic of the Butterworth filter is that it has a **gradual roll-off**. This means that it does not provide a sharp transition between the passband and stopband. Therefore, it may allow some lower frequency components to leak through, particularly close to the cutoff frequency.

3.2 K-means

- K-means clustering is a popular **unsupervised machine learning** algorithm used for clustering and grouping data points based on their similarity. The objective of the algorithm is to partition a given dataset into **K clusters**, where K is a user-defined parameter. Each cluster represents **a group of data points that are similar to each other** in terms of their features.

- The algorithm works as follows:

  Initialization: Randomly select K points from the dataset as initial **cluster centroids**.

  Assignment: Assign each data point to the nearest centroid based on a distance metric, often using **Euclidean distance**. This step creates initial clusters.

Update: Recalculate the centroid of each cluster by taking the **mean of all the data points** assigned to that cluster.

Iteration: Repeat steps 2 and 3 until convergence. Convergence occurs when the centroids no longer change significantly or when a maximum number of iterations is reached.

Final Clustering: Once convergence is achieved, the algorithm produces K clusters, where each data point belongs to the cluster with the nearest centroid.

3.3 STFT(Black-man window)

- **STFT** (Short-Time Fourier Transform) is a widely used time-frequency analysis technique in signal processing and audio analysis. It allows the examination of signal characteristics in **both the time and frequency** domains simultaneously by applying the Fourier transform to **short overlapping segments of the signal**.
- The STFT divides a **longer time-domain signal** into **shorter segments**, often referred to as **windows** or frames. Each windowed segment is then **Fourier transformed** to obtain its **frequency-domain representation**. By using a sliding window across the signal, the STFT provides a **time-varying** representation of the signal's frequency content.
- The windowing function plays a crucial role in the STFT as it affects the trade-off between time and frequency resolution. The **Blackman window** is a type of windowing function that helps **reduce spectral** leakage, which is a phenomenon where the **energy of a signal leaks into neighboring frequency bins** in the frequency domain. The spectral leakage can distort the frequency content of the signal and introduce unwanted artifacts.
- The Blackman window is designed to minimize spectral leakage by providing a **smooth tapering of the signal at the edges of each windowed segment.** It has a main lobe with low side lobes, which helps reduce the impact of spectral leakage.
- The formula for the Blackman window is given by:

w(n) = 0.42 - 0.5 cos $\left(\frac{2\pi n}{N-1}\right) + 0.08 \cos\left(\frac{4\pi n}{N-1}\right)$ (1)

where w(n) represents the value of the window at index n, and N is the length of the window.

- By applying the Blackman window to each segment before performing the Fourier transform in the STFT, it helps to mitigate spectral leakage and **improves the frequency resolution**. This allows for a more accurate representation of the signal's frequency content, especially **for narrowband signals** or signals with sharp spectral features.

3.4 Gated Recurrent Unit (GRU)

- GRU (Gated Recurrent Unit) is a type of recurrent neural network (RNN) architecture that is widely used for modeling **sequential data**. It is an extension of the more traditional LSTM (Long Short-Term Memory) network and is designed to address some of the limitations of LSTM while maintaining its ability to capture long-term dependencies in sequences.

- GRU introduces the concept of **"gates"** to control the flow of information within the network. These gates **regulate the information to be remembered, forgotten, or updated** in each time step. The GRU has two main gates: the reset gate and the update gate.

- Reset Gate: The reset gate determines **which information from the previous time step should be forgotten or ignored**. It takes into account the previous hidden state and the current input to selectively reset the hidden state.

- Update Gate: The update gate controls **how much information from the current time step should be incorporated into the hidden state**. It selectively updates the hidden state by combining the previous hidden state and the current input.

- The GRU architecture enables the model to adaptively decide whether to retain or update information based on the input and previous hidden state. This gating mechanism allows for efficient training and learning of temporal dependencies in sequential data.

4 Classifiers

4.1 Support Vector Machine (SVM)-Linear

- SVM (Support Vector Machine) is a **supervised machine learning algorithm** that can be used for both classification and regression tasks. The SVM-linear classifier specifically refers to the linear variant of SVM used for classification.

- The goal of the SVM-linear classifier is to **find an optimal hyperplane that separates the data points** belonging to different classes in a **linearly separable dataset.** The hyperplane is a decision boundary that maximizes the margin, which is the distance between the hyperplane and the closest data points from each class. SVM aims to find the hyperplane that **achieves the maximum margin while minimizing classification errors.**

- The key idea behind SVM-linear classifier is to transform the original input data into a **higher-dimensional feature space using a kernel function**. In this higher-dimensional space, **the data points become more separable by a hyperplane.** However, the SVM-linear classifier uses a linear kernel, which implicitly represents the data in the original input space without explicitly mapping it to a higher-dimensional feature space. This linear kernel simplifies the computation and allows the SVM to operate efficiently, even in high-dimensional spaces.

- To find the optimal hyperplane, the SVM-linear classifier solves an optimization problem that involves minimizing a cost function, often referred to as the **hinge loss** and maximizing the margin. The hinge loss **penalizes misclassified data points**, while the margin ensures good generalization by maximizing the separation between classes.

4.2 Support Vector Machine (SVM)-Radial Basis Function(RBF)

- SVM-RBF (Support Vector Machine with Radial Basis Function kernel) is an extension of the SVM algorithm that allows for **non-linear classification**.The RBF kernel is defined as a similarity measure between data points in a high-dimensional feature space. It transforms the data into a space where it **becomes more separable by a hyperplane.** The RBF kernel calculates the similarity between two data points based on the **Euclidean distance** between them. It **assigns higher similarity values** to points that are **closer together** and lower values to points that are farther apart.

- In SVM-RBF, the RBF kernel replaces the dot product used in the linear SVM. The kernel function computes the similarity between pairs of data points, and these similarities are used to define the decision boundary. By using the RBF kernel, SVM-RBF can capture non-linear relationships and **identify complex decision boundaries** that are necessary for handling more intricate classification tasks.

- The SVM-RBF algorithm aims to find the **optimal hyperplane that separates the data points** in a way that maximizes the margin and minimizes classification errors. It achieves this by **iteratively adjusting the position** and width of the decision boundary based on the support vectors, which are the data points closest to the decision boundary.

4.3 K-Nearest Neighbours(KNN)

- KNN (K-Nearest Neighbors) is a popular **supervised machine learning algorithm** used for both classification and regression tasks.The fundamental idea behind KNN is that **similar data points tend to belong to the same class** or have similar output values. In the case of classification, KNN determines the class label of a new data point by considering the class labels of its K nearest neighbors in the training set. The prediction is based on the **majority class among the K nearest neighbors**. In regression, KNN calculates the average or **weighted average of the output values of its K nearest neighbors** to predict the output value for the new data point.

- The K in KNN represents the number of neighbors considered for prediction. It is a hyperparameter that needs to be defined by the user. Choosing an appropriate value for K is important, as a **small K may lead to overfitting**, while a **large K may introduce more bias** into the predictions.

- The distance metric plays a crucial role in KNN for measuring the similarity between data points. Common **distance metrics** include **Euclidean distance, Manhattan distance, and Minkowski distance.** The choice of distance metric depends on the nature of the data and the problem at hand.

4.4 Decision Trees

- Decision trees are **supervised machine learning algorithms** that can be used for both classification and regression tasks.The structure of a decision tree resembles an upside-down tree, where each **internal node represents a decision based on a feature**, and each **leaf node represents a predicted outcome or class label.** The decision at each internal node is made by evaluating a specific feature, and the path followed through the tree leads to the final prediction at a leaf node.

- The decision tree learning process involves **recursively splitting the data based on features** to create nodes and determine the optimal decision boundaries. The goal is to find the **best feature and the best splitting criteria** at each node, which maximizes the information gain or minimizes the impurity in the resulting subsets.

5 Implementation and Methodology

The entire code has been divided and presented in subsections for easy understanding. The overall problem statement had four tracks,out of which the Track 1 and Track 4 use different datasets and have prediction tasks.While the track 2 and track3 use the same dataset and preprocessing as the track 1.

5.1 Track 1

5.1.1 Data Extraction

The dataset had 25 channels out of which 14 channels numbered from 4 to 17 were the eeg channels.They were plotted and those with no signal or noisy signals were disregarded.and finally on ly the 7 channels(F7 , F3 , P7 , O1 , O2 , P8 , AF4) were used.Then data was extracted from these channels in focus,unfocus and drowsy named dictionaries having (0-10 min),(10-20 min),(20-30 min) data of the experiment. The first 2 days of each subject were ignored thinking that the subject was not used to the experiment and the data after the first two days would be more relevant. Finally each state above had 5 days data for first 3 subjects and 4 days for next 2 subjects(as the dataset had only 6 days data for the last 2 subjects while for the first 3 subjects we had 7 days data). The following days belonged to each user: Subject 1: 3,4,5,6,7(1 and 2 ignored) Subject 2:10, 11, 12, 13, 14(9 and 8 ignored) Subject 3: 17,18,19,20,21(15 and 16 ignored) Subject 4: 24,25,26,27(22 ,23 and 28 ignored) Subject 5 :31,32,33,34(29,30,35 ignored) Thus ,Overall each state had 23 files of eeg data.

```python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import scipy.io
import matplotlib.pyplot as plt
import os
from scipy import signal
from scipy.fft import fft, fftshift



file_names=[]
for dirname, _, filenames in os.walk('/content/drive/MyDrive/EEG Data/
    EEG Data'):
    for filename in filenames:
        file_names.append(os.path.join(dirname, filename))
        #print(os.path.join(dirname, filename))

#print(file_names)
# each trial is about 54 mins
```

```python
import numpy as np
from scipy.fft import fft
from sklearn.cluster import KMeans
from google.colab import drive
drive.mount('/content/drive')


#starting from Channel 3,/0,1,2,3

# I have to plot all the 14 channels first.
#There are 14 channels, but the experimenter modified the headset.
#after removing the DC, the EEG amplitude should be around (-100,100).
#According to the plot, as the channels other than ['F7','F3','P7','O1
    ','O2','P8','AF4'] , don't have useful data thus only these are
    useful channels.
fig, ax = plt.subplots(14,1)
fig.set_figwidth(20)
fig.set_figheight(50)
#fig.set_size_inches(10,10)
mat = scipy.io.loadmat(file_names[0])
data = mat['o']['data'][0,0]
FS = mat['o']['sampFreq'][0][0][0][0]
channels = ['AF3', 'F7', 'F3', 'FC5', 'T7', 'P7', 'O1', 'O2', 'P8', 'T8'
    , 'FC6', 'F4', 'F8', 'AF4']
for i in range(14):
    data_ave = data[5000:15000,i+3]-np.mean(data[5000:15000,i+3])
    ax[i].plot(data_ave)
    ax[i].set_title(channels[i])
    ax[i].set_ylim(-100,100)


#F7,F3,P7,O1,O2,P8,AF4
useful_channels=[4,5,8,9,10,11,16]
useful_channels_names=[ 'F7', 'F3', 'P7', 'O1', 'O2', 'P8', 'AF4']
fig,ax = plt.subplots(8)
fig.set_size_inches(20,40)
j=0
for i in useful_channels:
    data_ave = data[5000:15000,i]-np.mean(data[5000:15000,i])
    ax[j].plot(data_ave)
    ax[j].set_ylim(-200,200)
    ax[j].set_title(channels[i-3])
    j=j+1


marker=128*60*10
#marker here is determinig the 10 minutes interval
```

```python
#delete file #28 because it doesnot have enough data
useful_file_index =
    [3,4,5,6,7,10,11,12,13,14,17,18,19,20,21,24,25,26,27,31,32,33,34]
#useful_file_index = np.arange(1,35)
chan_num=7
trail_names=[]
data_focus={}
data_unfocus={}
data_drowsy={}
focus={}
unfocus={}
drowsy={}
#for i in useful_file_index:
i=1
# here after enumerating the fimenames and getting their no in file name
     say 25 from eeg_record25.mat
#by splitting along the d and the .
# then by specifying the markers we divide the data in 0-10,10-20,20-30
    as focused ,unfocused and drowsy state
for index,filename in enumerate(filenames):
    if int(filename.split('d')[1].split('.')[0]) in useful_file_index:
        mat = scipy.io.loadmat(file_names[index])
        trail_names.append(filename.split('.')[0])
        data_focus[trail_names[-1]]=mat['o']['data'][0,0][0:marker,
    useful_channels].copy()
        data_unfocus[trail_names[-1]]=mat['o']['data'][0,0][marker:2*
    marker,useful_channels].copy()
        data_drowsy[trail_names[-1]]=mat['o']['data'][0,0][2*marker:3*
    marker,useful_channels].copy()
        focus[trail_names[-1]]=mat['o']['data'][0,0][0:marker,
    useful_channels].copy()
        unfocus[trail_names[-1]]=mat['o']['data'][0,0][marker:2*marker,
    useful_channels].copy()
        drowsy[trail_names[-1]]=mat['o']['data'][0,0][2*marker:3*marker,
    useful_channels].copy()
```

5.1.2 Filtering

The data was then passed through a butterman highpass filter of 5th order and 0.16Hz cut-off frequency.

```python
# build 5 order high pass filter
from scipy.signal import butter, lfilter, freqz
# ----- ----- ----- -----
def butter_highpass(cutoff, fs, order=5):
    nyq = 0.5 * fs
    normal_cutoff = cutoff / nyq
    b, a = signal.butter(order, normal_cutoff, btype='high', analog=
   False)
    return b, a

def butter_highpass_filter(data, cutoff, fs, order=5):
    b, a = butter_highpass(cutoff, fs, order=order)
    x = signal.filtfilt(b, a, data)
    y = signal.filtfilt(b, a, x)
    return y

# High Pass 0.16HZ
row, col = data_focus['eeg_record3'].shape
for name in trail_names:
    for i in range(col):
        data_focus[name][:,i]=butter_highpass_filter(data_focus[name][:,
   i], 0.16, 128, 5)
        data_unfocus[name][:,i]=butter_highpass_filter(data_unfocus[name
   ][:,i], 0.16, 128, 5)
        data_drowsy[name][:,i]=butter_highpass_filter(data_drowsy[name
   ][:,i], 0.16, 128, 5)
        #print(name,data_drowsy[name][:,i].shape)
```

5.1.3 Feature Extraction

The following methodology was applied in the feature extraction.

Defining Frequency Bins and Feature Names: A frequency range from 0.5 Hz to 18 Hz is divided into 0.5 Hz bins. Feature names are created by combining the channel names with the corresponding frequency bin using an underscore .

Validating High-Pass Filter: Two plots are generated to validate the high-pass filter. The first plot shows the raw signal and the filtered signal for the 'eeg_record3' data. The second plot shows the same for the 'eeg_record33' data.

Calculating Short-Time Fourier Transform (STFT): The Short-Time Fourier Transform (STFT) is calculated for each EEG channel using a Blackman window of size 128. The STFT is computed at 1-second intervals within each input EEG channel, resulting in a set of time-varying Discrete Fourier Transform (DFT) amplitudes.

Calculating Power and Averaging: Power is calculated by taking the squared absolute values of the STFT. Power values are then averaged over 4 consecutive frequency bins (0.5 Hz each) and 15-second running windows to reduce dimensionality and improve temporal resolution.

Constructing SVM Vectors: The power values are transformed into SVM vectors by reshaping each channel's data at a specific frequency into a single vector. The resulting SVM vectors are logarithmically scaled by applying a logarithmic transformation.

Visualizing Channel-Frequency Relationships: Plots are generated to visualize the relationship between EEG channels and specific frequency components. The plots show how each channel changes with time for a particular frequency and how each frequency component changes with frequency for a specific channel.

```
feature_names = []
freq_range=np.arange(0.5,18.5,0.5)
symb='_'
#useful_channels_names=['F7','F3','P7','O1','O2','P8','AF4']
for index,channel in enumerate(useful_channels_names):
    for freq in freq_range:
        feature_names.append(channel+symb+str(freq))
feature_names


#validate the high pass filter
fig,ax = plt.subplots(1)
fig.set_size_inches(20,10)
color = 'tab:red'
ax.set_xlabel('X-axis')
ax.set_ylabel('Y1-axis', color = color)
ax.plot(unfocus['eeg_record3'][0:500,0], color = color,label='raw signal
    ')
ax.tick_params(axis ='y', labelcolor = color)
ax.set_ylim(3500,4100)
# Adding Twin Axes to plot using dataset_2
ax0 = ax.twinx()
color = 'tab:green'
ax0.plot(data_unfocus['eeg_record3'][0:500,0],color=color,label='
    filtered signal')
ax0.set_ylim(-150,150)
fig.legend()
plt.show()

#validate the high pass filter
fig,ax = plt.subplots(1)
fig.set_size_inches(20,10)
color = 'tab:red'
ax.set_xlabel('X-axis')
ax.set_ylabel('Y1-axis', color = color)
ax.plot(unfocus['eeg_record33'][0:500,0], color = color,label='raw
    signal')
ax.tick_params(axis ='y', labelcolor = color)
ax.set_ylim(3800,4200)
# Adding Twin Axes to plot using dataset_2
ax0 = ax.twinx()
color = 'tab:green'
```

```python
ax0.plot(data_unfocus['eeg_record33'][0:500,0],color=color,label='filtered signal')
ax0.set_ylim(-100,100)
fig.legend()
plt.show()

# STFT was then calculated at a time step of 1 s producing a set of time-varying DFT
# amplitudes X STFT (t,  ) at 1s intervals within each input EEG channel.
t_win = np.arange(0,128)
M = 128
window_blackman=0.42-0.5*np.cos((2*np.pi*t_win)/(M-1))+0.08*np.cos((4*np.pi*t_win)/(M-1))#window_blackman = signal.windows.blackmanharris(128)

#col is 7
power_focus = {}
for name in trail_names:
    power_focus[name]=np.zeros([col,513,601])

power_unfocus = {}
for name in trail_names:
    power_unfocus[name]=np.zeros([col,513,601])

power_drowsy = {}
for name in trail_names:
    power_drowsy[name]=np.zeros([col,513,601])

#the output of the stft is 513*601,1 second data will produce 1 column of data,there are 601
for name in trail_names:
    for i in range(col):
        f, t,y1=scipy.signal.stft(data_focus[name][:,i],fs=128, window=window_blackman, nperseg=128,
                      noverlap=0, nfft=1024, detrend=False, return_onesided=True, boundary='zeros',
                      padded=True)
        f, t,y2=scipy.signal.stft(data_unfocus[name][:,i],fs=128, window=window_blackman, nperseg=128,
                      noverlap=0, nfft=1024, detrend=False, return_onesided=True, boundary='zeros',
                      padded=True)
        f, t,y3=scipy.signal.stft(data_drowsy[name][:,i],fs=128, window=window_blackman, nperseg=128,
                      noverlap=0, nfft=1024, detrend=False, return_onesided=True, boundary='zeros',
                      padded=True)
        power_focus[name][i,:,:]=(np.abs(y1))**2
        power_unfocus[name][i,:,:]=(np.abs(y2))**2
        power_drowsy[name][i,:,:]=(np.abs(y3))**2

#validate the power value
fig,ax = plt.subplots(col,1)
fig.set_size_inches(20,50)
for i in range(col):
    ax[i].pcolormesh(t, f, power_focus['eeg_record18'][i,:,:],vmin=0, vmax=2*np.sqrt(2), shading='gouraud')

fig,ax = plt.subplots(col,1)
fig.set_size_inches(20,50)
for i in range(col):
    ax[i].pcolormesh(t, f, power_focus['eeg_record33'][i,:,:],vmin=0, vmax=2*np.sqrt(2), shading='gouraud')


# combine bins into 0.5HZ, and keep 0-18 HZ.

num=[]

power_focus_bin = {}
for name in trail_names:
    power_focus_bin[name]=np.zeros([7,36,601])

power_unfocus_bin = {}
for name in trail_names:
    power_unfocus_bin[name]=np.zeros([7,36,601])

power_drowsy_bin = {}
for name in trail_names:
    power_drowsy_bin[name]=np.zeros([7,36,601])

for name in trail_names:
    for chn in range(col):
        j=0
        for i in range(1,144,4):
            power_focus_bin[name][chn,j,:]=np.average(power_focus[name][chn,i:i+4,:],axis=0)
            power_unfocus_bin[name][chn,j,:]=np.average(power_unfocus[name][chn,i:i+4,:],axis=0)
            power_drowsy_bin[name][chn,j,:]=np.average(power_drowsy[name][chn,i:i+4,:],axis=0)
            #print(np.average(power_drowsy[name][chn,i:i+4,:],axis=0).shape)
            #if name=='eeg_record3':
            #    if chn==0:
            #        num.append((f[i:i+4]))
            #    print(j)
            j=j+1


#print(num)
#print(len(num))

# avarage over 15 seconds running window.

power_focus_ave = {}
for name in trail_names:
    power_focus_ave[name]=np.zeros([7,36,585])

power_unfocus_ave = {}
for name in trail_names:
    power_unfocus_ave[name]=np.zeros([7,36,585])

power_drowsy_ave = {}
for name in trail_names:
    power_drowsy_ave[name]=np.zeros([7,36,585])

for name in trail_names:
    for chn in range(col):
        j=0
        for k in range(0,585):
            power_focus_ave[name][chn,:,j]=np.average(power_focus_bin[name][chn,:,k:k+15],axis=1)
            power_unfocus_ave[name][chn,:,j]=np.average(power_unfocus_bin[name][chn,:,k:k+15],axis=1)
            power_drowsy_ave[name][chn,:,j]=np.average(power_drowsy_bin[name][chn,:,k:k+15],axis=1)
            #print(np.average(power_drowsy_bin[name][chn,:,k:k+15],axis=1).shape)
            j=j+1


# Turn the data into a vector
#[252,585]

svm_focus = {}
for name in trail_names:
    svm_focus[name]=np.zeros([252,585])

svm_unfocus = {}
```

```python
for name in trail_names:
    svm_unfocus[name]=np.zeros([252,585])

svm_drowsy = {}
for name in trail_names:
    svm_drowsy[name]=np.zeros([252,585])

for name in trail_names:
    for j in range(585):
        svm_focus[name][:,j]=power_focus_ave[name][:,:,j].reshape(1,-1)
        svm_unfocus[name][:,j]=power_unfocus_ave[name][:,:,j].reshape
    (1,-1)
        svm_drowsy[name][:,j]=power_drowsy_ave[name][:,:,j].reshape
    (1,-1)
    svm_focus[name]=10*np.log(svm_focus[name])
    svm_unfocus[name]=10*np.log(svm_unfocus[name])
    svm_drowsy[name]=10*np.log(svm_drowsy[name])
# now, we get the svm vector 252*585 252 rows


# along the horizontal direction ,it shows one channel at specific
    frequecy changes with time.
# each channel has 36 rows of data,
# the following shows 7 channels, each with 36 slots of frequency.
fig,ax = plt.subplots(36,1)
fig.set_size_inches(10,150)
for i in range(36):
    ax[i].plot(svm_focus['eeg_record33'][i,:],label='focus')
    ax[i].plot(svm_drowsy['eeg_record33'][i,:],c='r',label='drowsy')
    ax[i].plot(svm_unfocus['eeg_record33'][i,:],c='green',label='unfocus
    ')
    ax[i].legend()
    ax[i].set_title("The first channel with Frequency at: "+ str((i+1)
    *0.5) )

# along the horizontal direction ,it shows one channel at specific
    frequecy changes with time.
# each channel has 36 rows of data,
# the following shows 7 channels, each with 36 slots of frequency.
fig,ax = plt.subplots(36,1)
fig.set_size_inches(10,150)
for i in range(36):
    ax[i].plot(svm_focus['eeg_record5'][i,:],label='focus')
    ax[i].plot(svm_drowsy['eeg_record5'][i,:],c='r',label='drowsy')
    ax[i].plot(svm_unfocus['eeg_record5'][i,:],c='green',label='unfocus'
    )
    ax[i].legend()
    ax[i].set_title("The first channel with Frequency at: "+ str((i+1)
   *0.5) )

# along the horizontal direction ,it shows one channel at specific
    frequecy changes with time.
# each channel has 36 rows of data,
# the following shows 7 channels, each with 36 slots of frequency.
fig,ax = plt.subplots(36,1)
fig.set_size_inches(10,150)
for i in range(36):
    ax[i].plot(svm_focus['eeg_record5'][i,:],label='focus')
    ax[i].plot(svm_drowsy['eeg_record5'][i,:],c='r',label='drowsy')
    ax[i].plot(svm_unfocus['eeg_record5'][i,:],c='green',label='unfocus'
   )
    ax[i].legend()
    ax[i].set_title("The first channel with Frequency at: "+ str((i+1)
   *0.5) )

fig,ax = plt.subplots(36,1)
fig.set_size_inches(10,100)
#at a specific time, each channel changes with frequency
for i in range(36):
    ax[i].plot(svm_focus['eeg_record33'][:,i],label='focus')
    ax[i].plot(svm_drowsy['eeg_record33'][:,i],c='r',label='drowsy')
    ax[i].plot(svm_unfocus['eeg_record33'][:,i],c='green',label='unfocus
   ')
    ax[i].legend()
    ax[i].set_title("The first channel with Frequency at: "+ str((i+1)
   *0.5) )
# the peak is at the lowest frequency.
#Each peak is the start of the data.

#at a specific time, each channel changes with frequency
for i in range(0,25,5):
    plt.plot(svm_focus[trail_names[i]][:,1])
# the peak is at the lowest frequency.
#Each peak is the start of the data.
plt.title("The first channel with Frequency at: "+ str((21+1)*0.5))
plt.show()
```

5.1.4 Classifier Implementations

The following methodology is being followed in the below code: **Creating Target Labels:** Target labels are created for the three cognitive states: 'focus', 'unfocus', and 'drowsy'. Three lists are created: 'label_focus' containing 0s (representing the 'focus' state), 'label_unfocus' containing 1s (representing the 'unfocus' state), and 'label_drowsy' containing 2s (representing the 'drowsy' state).

Data Preparation for Multiple Subjects: EEG data and target labels are prepared for multiple subjects. Separate variables are created for each subject's data and target labels (e.g., subj1, subj2, subj3, subj4, subj5).

Training Support Vector Machine (SVM) Model for Subject 2: The SVM model is trained using the data and target labels from subject 2. The data is split into training and testing sets using the 'train_test_split' function. The training data is then scaled using 'StandardScaler'. The SVM model with a radial basis function (RBF) kernel is trained and evaluated using the scaled training and testing data. The training and testing accuracy scores are printed.

Training SVM Models for All Subjects: The SVM models are trained using the data and target labels from all subjects. The data is split into training and testing sets using 'train_test_split'. The training data is scaled using 'StandardScaler'. Two SVM models are trained and evaluated: one with a linear kernel and the other with an RBF kernel. The training and testing accuracy scores are printed for each model.

Principal Component Analysis (PCA): PCA is applied to the scaled training data. The variance explained by each principal component is calculated, and a heatmap is created to visualize the correlation between features in the training data.

Training SVM Model with PCA: The SVM model with an RBF kernel is trained using the PCA-transformed training data. The training and testing accuracy scores are printed.

Training K-Nearest Neighbors (KNN) Model: The KNN model is trained using the scaled training data. The training and testing accuracy scores are printed.

Training Decision Trees Model: The decision tree classifier is trained using the scaled training data. The training and testing accuracy scores are printed.

Training Random Forest Classifier Model: The random forest classifier is trained using the scaled training data. The accuracy scores are calculated for different values of the maximum depth of the decision trees in the random forest, and a plot is created to visualize the relationship between depth and accuracy.

Training SVM Model with K-Fold Cross-Validation: The SVM model with an RBF kernel is trained using K-fold cross-validation. The training and cross-validation accuracy scores are printed for each fold, and the average scores are calculated.

```
#--------0
label_focus = [0]*585
#--------1
label_unfocus = [1]*585
#--------2
label_drowsy = [2]*585

```

```python
#subject is the variable for all participants

subj1_files={'eeg_record3','eeg_record4','eeg_record5','eeg_record6','
    eeg_record7'}
subj2_files={'eeg_record10','eeg_record11','eeg_record12','eeg_record13'
    ,'eeg_record14'}
subj3_files={'eeg_record17','eeg_record18','eeg_record19','eeg_record20'
    ,'eeg_record21'}
subj4_files={'eeg_record24','eeg_record25','eeg_record26','eeg_record27'
    }
subj5_files={'eeg_record31','eeg_record32','eeg_record33','eeg_record34'
    }


# I will try to use the data from all participants to train the model
target=[]
subj=np.array([]).reshape(252,0).copy()
for name in trail_names:
    subj=np.concatenate((subj,svm_focus[name]), axis=1)
    subj=np.concatenate((subj,svm_unfocus[name]), axis=1)
    subj=np.concatenate((subj,svm_drowsy[name]), axis=1)
    target = target+label_focus+label_unfocus+label_drowsy
subj=subj.T
target = np.array(target)

# This part I only train the data for subject1
target1=[]
subj1=np.array([]).reshape(252,0).copy()
for name in subj1_files:
    subj1=np.concatenate((subj1,svm_focus[name]), axis=1)
    subj1=np.concatenate((subj1,svm_unfocus[name]), axis=1)
    subj1=np.concatenate((subj1,svm_drowsy[name]), axis=1)
    target1 = target1+label_focus+label_unfocus+label_drowsy
subj1=subj1.T
target1 = np.array(target1)

target2=[]
subj2=np.array([]).reshape(252,0).copy()
for name in subj2_files:
    subj2=np.concatenate((subj2,svm_focus[name]), axis=1)
    subj2=np.concatenate((subj2,svm_unfocus[name]), axis=1)
    subj2=np.concatenate((subj2,svm_drowsy[name]), axis=1)
    target2 = target2+label_focus+label_unfocus+label_drowsy
subj2=subj2.T
target2 = np.array(target2)


```

```
target3=[]
subj3=np.array([]).reshape(252,0).copy()
for name in subj3_files:
    subj3=np.concatenate((subj3,svm_focus[name]), axis=1)
    subj3=np.concatenate((subj3,svm_unfocus[name]), axis=1)
    subj3=np.concatenate((subj3,svm_drowsy[name]), axis=1)
    target3 = target3+label_focus+label_unfocus+label_drowsy
subj3=subj3.T
target3 = np.array(target3)

target4=[]
subj4=np.array([]).reshape(252,0).copy()
for name in subj4_files:
    subj4=np.concatenate((subj4,svm_focus[name]), axis=1)
    subj4=np.concatenate((subj4,svm_unfocus[name]), axis=1)
    subj4=np.concatenate((subj4,svm_drowsy[name]), axis=1)
    target4 = target4+label_focus+label_unfocus+label_drowsy
subj4=subj4.T
target4 = np.array(target4)

target5=[]
subj5=np.array([]).reshape(252,0).copy()
for name in subj5_files:
    subj5=np.concatenate((subj5,svm_focus[name]), axis=1)
    subj5=np.concatenate((subj5,svm_unfocus[name]), axis=1)
    subj5=np.concatenate((subj5,svm_drowsy[name]), axis=1)
    target5 = target5+label_focus+label_unfocus+label_drowsy
subj5=subj5.T
target5 = np.array(target5)

print('length of the target:',len(target))
print('the shape of the data from the subject1:', subj.shape)

# Train the data from subject1
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.svm import SVC

data_train, data_test, data_train_target, data_test_target =
    train_test_split(subj5, target5, test_size=0.8, random_state=0)
scaler = preprocessing.StandardScaler().fit(data_train)
X_train_scaled = scaler.transform(data_train)
X_test_scaled = scaler.transform(data_test)

svm = SVC(kernel='rbf')
svm.fit(X_train_scaled,data_train_target)
```

```python
print(f'The Score for Training data with SVM Model for subject2:',svm.
    score(X_train_scaled,data_train_target))
print(f'Score of For Test data with SVM Model for subject2 : {svm.score(
    X_test_scaled,data_test_target)}')

# Use 60% data for test
from sklearn.decomposition import PCA
data_train, data_test, data_train_target, data_test_target =
    train_test_split(subj, target, test_size=0.6, random_state=0)
scaler = preprocessing.StandardScaler().fit(data_train)
X_train_scaled = scaler.transform(data_train)
X_test_scaled = scaler.transform(data_test)
#PCA should be used on scaled data
pca = PCA()
pca.fit(X_train_scaled)
X_train_pca = pca.transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)
var = pca.explained_variance_/pca.explained_variance_.sum()
var[0:20].sum()

import seaborn as sns

data_corr = pd.DataFrame(X_train_scaled).corr()
sns.heatmap(data_corr)
plt.show()

import seaborn as sns
#sns.heatmap(pca.components_)
#print(sum(pca.components_[:,0]**2))

pca_map=pd.DataFrame(pca.components_,columns=feature_names,index=np.
    arange(1,253))
#pca.components_.shape
pca_map


#USE SVM linear model
svm = SVC(kernel='linear')
svm.fit(X_train_scaled,data_train_target)
print(f'The Score for Training data with SVM Linear Model for all
    subjects:',svm.score(X_train_scaled,data_train_target))
print(f'Score of For Test data with SVM Linear Model for all subjects :
    {svm.score(X_test_scaled,data_test_target)}')

#Use RBF model
svm = SVC(kernel='rbf')
svm.fit(X_train_scaled,data_train_target)
```

```
print(f'The Score for Training data with SVM RBF Model for all subjects:
    ',svm.score(X_train_scaled,data_train_target))
print(f'Score of For Test datawith SVM RBF Model for all subjects : {svm
    .score(X_test_scaled,data_test_target)}')

svm = SVC(kernel='rbf')
svm.fit(X_train_pca[:,0:30],data_train_target)
print(f'The Score for Training data with SVM RBF Model for all subjects:
    ',svm.score(X_train_pca[:,0:30],data_train_target))
print(f'Score of For Test data with SVM RBF Model for all subjects : {
    svm.score(X_test_pca[:,0:30],data_test_target)}')

#Try KNN Model
from sklearn.neighbors import KNeighborsClassifier
neighbor = KNeighborsClassifier(n_neighbors=3)
neighbor.fit(X_train_scaled,data_train_target)
print("the score for training with data from 5 participants:",neighbor.
    score(X_train_scaled,data_train_target))
print("the score for test data from 5 participants:",neighbor.score(
    X_test_scaled,data_test_target))

neighbor.fit(X_train_pca[:,0:30],data_train_target)
print("the score for training with data from 5 participants:",neighbor.
    score(X_train_pca[:,0:30],data_train_target))
print("the score for test data from 5 participants:",neighbor.score(
    X_test_pca[:,0:30],data_test_target))

# decision trees
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(max_depth=16)
dt.fit(X_train_scaled,data_train_target)
print(f"the score for training with data from 5 participants:{dt.score(
    X_train_scaled,data_train_target)}")
print("the score for test data from 5 participants:",dt.score(
    X_test_scaled,data_test_target))

from sklearn.ensemble import RandomForestClassifier

train_scores = []
test_scores = []
for depth in range(1,35):
     dt_reg = RandomForestClassifier(max_depth=depth,random_state=0)
     dt_reg.fit(X_train_scaled,data_train_target)
     train_scores = train_scores +[dt_reg.score(X_train_scaled,
    data_train_target)]
     test_scores = test_scores + [dt_reg.score(X_test_scaled,
    data_test_target)]
```

```

x = list(range(1,35))
plt.plot(x,train_scores,c='r',label='train')
plt.plot(x,test_scores,c='b',label='test')
plt.xlabel('depth')
plt.ylabel('accuracy')
plt.title('Depth vs. Accuracy for Random Forest Classifier')
plt.show()


#from sklearn import tree

# graphviz.Source takes a graphviz object and turns it into an image.
    Graphviz is a
# standard graphical library/format for rendering graphs - nodes
    connected by edges -
# such as our decision tree below.
#graphviz.Source(tree.export_graphviz(dt,feature_names=feature_names))
#r=tree.export_text(dt, feature_names=feature_names)
#print(r)

from sklearn.model_selection import KFold
import tensorflow as tf
tf.random.set_seed(0)
kfold = KFold(n_splits=5, shuffle=True)

data_train_target=data_train_target

fold_no = 1

score_tr=[]
score_cv=[]

svm = SVC(kernel='rbf')

for train,cv in kfold.split(X_train_scaled,data_train_target):
    X_tr = X_train_scaled[train]
    Y_tr = data_train_target[train]
    X_cv = X_train_scaled[cv]
    Y_cv = data_train_target[cv]

    tf.random.set_seed(0)

    svm.fit(X_tr,Y_tr)
    score_tr.append(svm.score(X_tr,Y_tr))
    score_cv.append(svm.score(X_cv,Y_cv))

    print(f'Score for {fold_no} Fold Training: {score_tr[-1]:.3f}')
    print(f'Score for {fold_no} Fold cv     : {score_cv[-1]:.3f}')
    print('-----------------------------------')
    fold_no = fold_no + 1

print(f'Score of Average For Training: {np.mean(score_tr):.3f}')
print(f'Score of Average For CV.      : {np.mean(score_cv):.3f}')
```

5.2 Track 2,3

The code here was used after basic filtering of the track 1 . In this same piece of code the variables of different subjects,their states,and indices of different eeg channels were used to generate a huge set of plots .Where each plot plotted the percentages of alpha,beta,theta waves corresponding to all the days of particular subject,with particular state,and specific channels. The methodology followed here was: **Defining Frequency Bands:** Frequency bands for brainwaves (theta, alpha, and beta) are defined using their corresponding frequency ranges.
Computing Power Spectral Density (PSD): The Fast Fourier Transform (FFT) is used to compute the power spectral density (PSD) of the EEG data samples. The PSD represents the power distribution across different frequency components.
Calculating Brainwave Powers: The power within each frequency band (theta, alpha, beta) is calculated by summing the PSD values within the corresponding frequency range.
Calculating Percentage of Brainwave Powers: The total power is calculated as the sum of theta, alpha, and beta powers. The percentages of theta, alpha, and beta powers are then computed by dividing each power by the total power and multiplying by 100.
Performing K-means Clustering: The calculated brainwave percentages (theta, alpha, beta) are combined into a feature vector. K-means clustering is applied to cluster the feature vectors into three clusters (assuming 'n-clusters=3').

```
import numpy as np
from scipy.fft import fft
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Assuming you have a list of 5 preprocessed EEG data samples stored in
    'eeg_data'
# 'eeg_data' should be a list of 1D numpy arrays

# Define frequency bands for brainwaves
frequency_bands = {
    'theta': (4, 8),     # Theta band (4 Hz - 8 Hz)
    'alpha': (8, 13),    # Alpha band (8 Hz - 13 Hz)
    'beta': (13, 18),    # Beta band (13 Hz - 18 Hz)
```

```python
}

# Initialize lists to store brainwave percentages for each iteration
theta_percentages = []
alpha_percentages = []
beta_percentages = []
eeg_data_files=[]

# EEG data file names
for i in range(5):

    eeg_data_files.append('eeg_record'+str(i+10))

# Iterate over each EEG data sample
for i in range(5):
    # Compute power spectral density (PSD) using FFT
    psd = np.abs(fft(data_drowsy[eeg_data_files[i]][:,[2,5]])) ** 2

    # Calculate power for theta, alpha, and beta brainwave frequencies
    theta_power = np.sum(psd[int(frequency_bands['theta'][0]):int(
   frequency_bands['theta'][1])])
    alpha_power = np.sum(psd[int(frequency_bands['alpha'][0]):int(
   frequency_bands['alpha'][1])])
    beta_power = np.sum(psd[int(frequency_bands['beta'][0]):int(
   frequency_bands['beta'][1])])

    # Calculate total power
    total_power = theta_power + alpha_power + beta_power

    # Calculate percentages of theta, alpha, and beta powers
    theta_percentage = (theta_power / total_power) * 100
    alpha_percentage = (alpha_power / total_power) * 100
    beta_percentage = (beta_power / total_power) * 100

    # Append the percentages to the respective lists
    theta_percentages.append(theta_percentage)
    alpha_percentages.append(alpha_percentage)
    beta_percentages.append(beta_percentage)

# Perform K-means clustering on the brainwave percentages
X = np.array(list(zip(theta_percentages, alpha_percentages,
   beta_percentages)))
kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(X)

# Plot the brainwave frequencies for each iteration
bar_width = 0.2
```

```
index = np.arange(1, 6)

plt.bar(index - bar_width, theta_percentages, width=bar_width, label='
    Theta')
plt.bar(index, alpha_percentages, width=bar_width, label='Alpha')
plt.bar(index + bar_width, beta_percentages, width=bar_width, label='
    Beta')

plt.xlabel('Subject 2')
plt.ylabel('Percentage(Brainwave frequencies)')
plt.title('Brainwave Frequencies')

# Set y-axis labels with EEG data file names
plt.xticks(index, [f'day {i+10}' for i in range(5)])

plt.legend()
plt.show()
```

5.3 Track 4

The methodology followed in the track 4 was as follows:

Data Visualization:

The code begins by visualizing a sample of the features 'fft_0_b' through 'fft_749_b' using a line plot.

Data Preprocessing:

The target labels are mapped to numerical values (0, 1, 2) using a label mapping dictionary. The dataset is split into training and testing sets using the train_test_split function.

Model Architecture:

The model architecture is defined using the TensorFlow Keras API. The input shape is determined by the number of features in the dataset. A GRU (Gated Recurrent Unit) layer is added to capture temporal dependencies in the data. A flatten layer is used to convert the output of the GRU layer into a 1D tensor. The final output layer is a dense layer with softmax activation for multiclass classification.

Model Training:

The model is compiled with the Adam optimizer, sparse categorical cross-entropy loss, and accuracy metric. Early stopping is applied to monitor the validation loss and stop training if there is no improvement for a specified number of epochs. ModelCheckpoint saves the best model based on validation accuracy during training. LearningRateScheduler adjusts the learning rate during training. The model is trained on the training data with validation data specified for monitoring. Training history, including accuracy and loss, is stored for visualization.

Model Evaluation:

The model's accuracy is evaluated on the test data. The predicted labels are generated using the model on the test data. Confusion matrix and classification report are generated based on the predicted labels and actual labels of the test data. The confusion matrix is visualized using a heatmap. The classification report provides precision, recall, F1-score, and support for each class.

```python
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split

import tensorflow as tf
from tensorflow.keras.callbacks import EarlyStopping,
    LearningRateScheduler, ModelCheckpoint

from sklearn.metrics import confusion_matrix, classification_report


sample = data.loc[0, 'fft_0_b':'fft_749_b']

plt.figure(figsize=(16, 10))
plt.plot(range(len(sample)), sample)
plt.title("Features fft_0_b through fft_749_b")
plt.show()

data['label'].value_counts()

label_mapping = {'NEGATIVE': 0, 'NEUTRAL': 1, 'POSITIVE': 2}

def preprocess_inputs(df):
    df = df.copy()

    df['label'] = df['label'].replace(label_mapping)

    y = df['label'].copy()
    X = df.drop('label', axis=1).copy()

    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size
    =0.7, random_state=123)

    return X_train, X_test, y_train, y_test
```

```

X_train, X_test, y_train, y_test = preprocess_inputs(data)

inputs = tf.keras.Input(shape=(X_train.shape[1],))

expand_dims = tf.expand_dims(inputs, axis=2)

gru = tf.keras.layers.GRU(1024, return_sequences=True)(expand_dims)

flatten = tf.keras.layers.Flatten()(gru)

outputs = tf.keras.layers.Dense(3, activation='softmax')(flatten)


model = tf.keras.Model(inputs=inputs, outputs=outputs)
print(model.summary())

model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience
    =10, restore_best_weights=True)
mc = ModelCheckpoint('./best_dnn_model.h5', monitor='val_accuracy', mode
    ='max', verbose=1, save_best_only=True)
lr_schedule = LearningRateScheduler(lambda epoch: 0.001 * np.exp(-epoch
    / 10.))

history = model.fit(
    X_train,
    y_train,
    validation_split=0.2,
    validation_data=(X_test, y_test),
    batch_size=32,
    epochs=100,
    callbacks=[es, mc, lr_schedule]
)

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```python
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

model_acc = model.evaluate(X_test, y_test, verbose=0)[1]
print("Test Accuracy: {:.3f}%".format(model_acc * 100))

y_pred = np.array(list(map(lambda x: np.argmax(x), model.predict(X_test)
    )))

cm = confusion_matrix(y_test, y_pred)
clr = classification_report(y_test, y_pred, target_names=label_mapping.
    keys())

plt.figure(figsize=(8, 8))
sns.heatmap(cm, annot=True, vmin=0, fmt='g', cbar=False, cmap='Greens')
plt.xticks(np.arange(3) + 0.5, label_mapping.keys())
plt.yticks(np.arange(3) + 0.5, label_mapping.keys())
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

print("Classification Report:\n----------------------\n", clr)
```

6 Results and Conclusion

6.1 Track 1:Cognitive state prediction

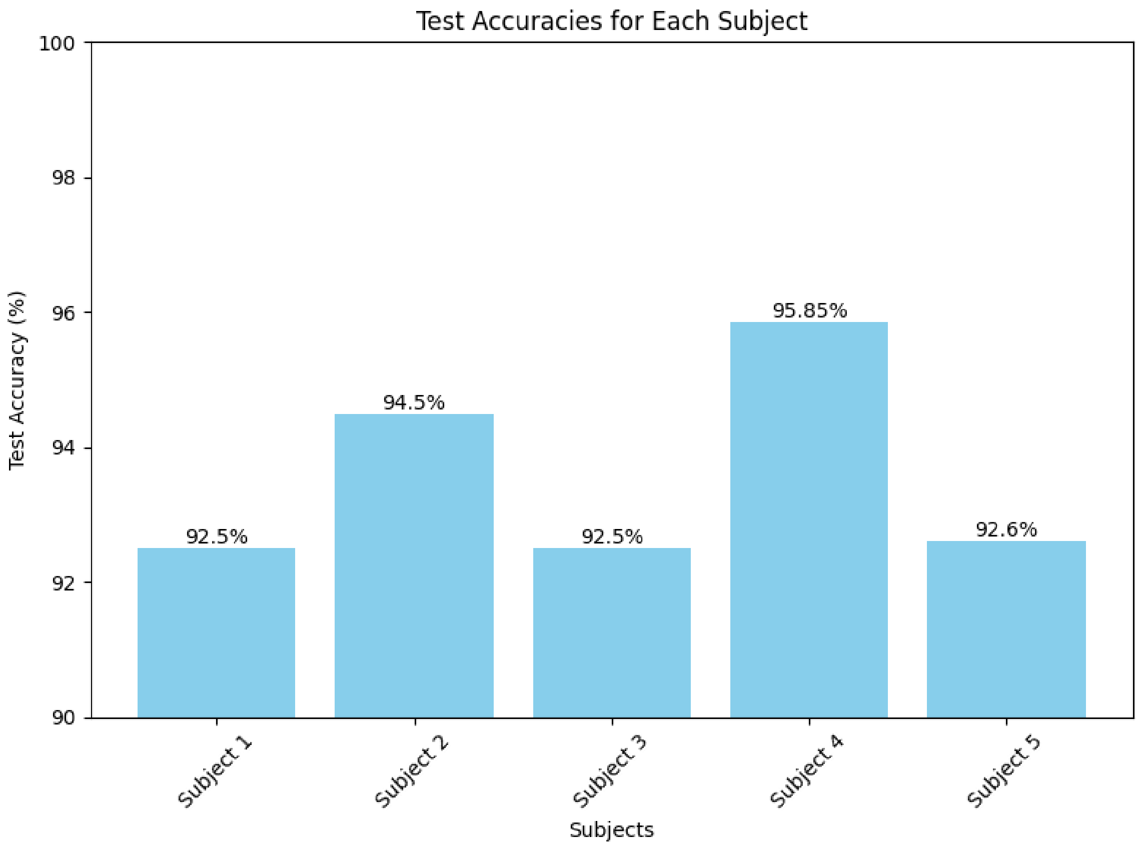

Figure 1: Test accuracies of SVM-linear on training and testing on each subject individually

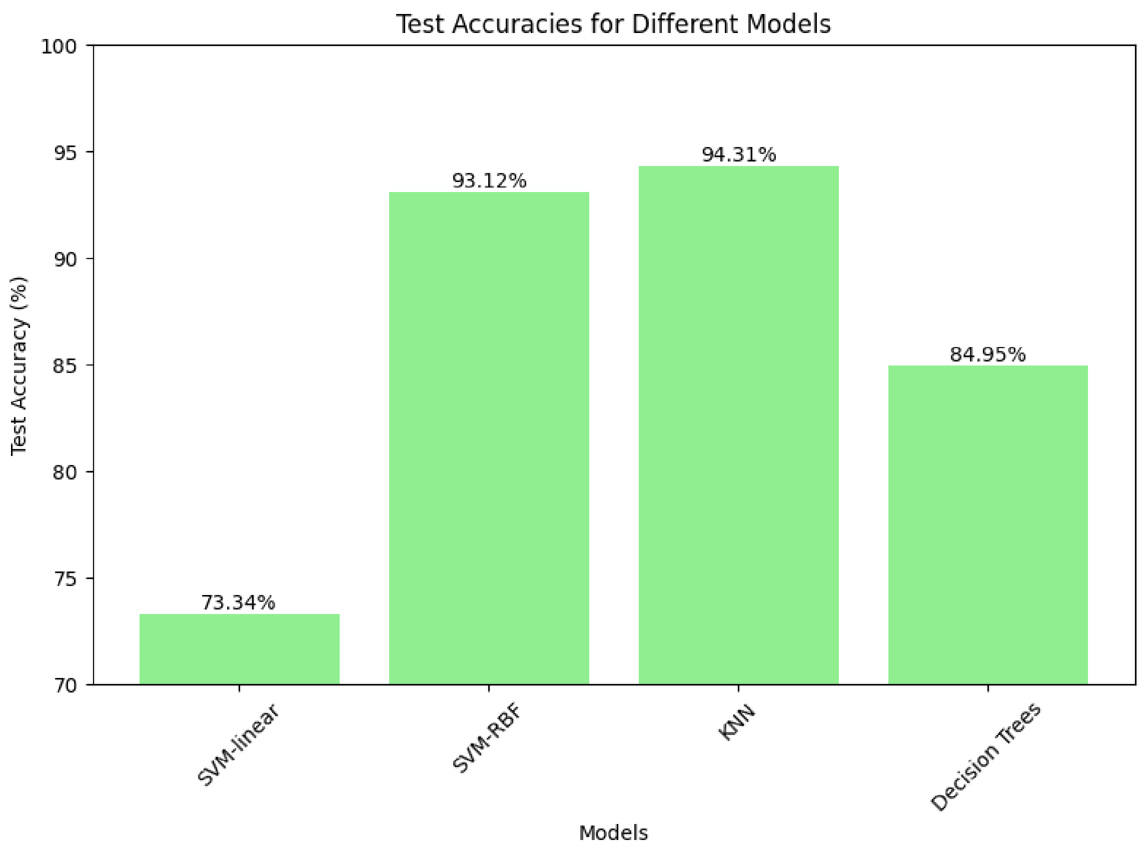

Figure 2: Test accuracies of various models trained and tested on all-subject combined data

6.2 Track 2:Spectral presence on brainwave frequencies in different cognitive states

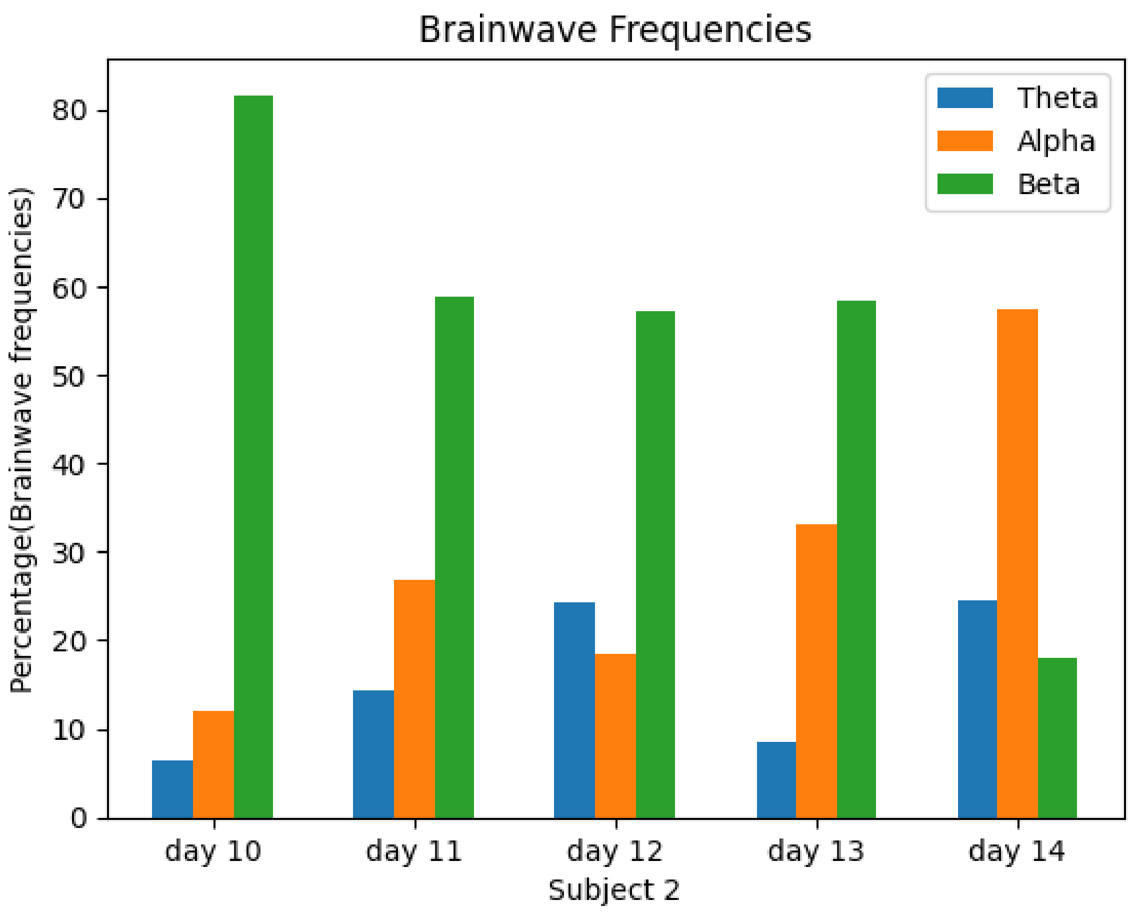

Figure 3: Percentage of brainwave frequencies in **focussed state** of Subject 2 using all channels

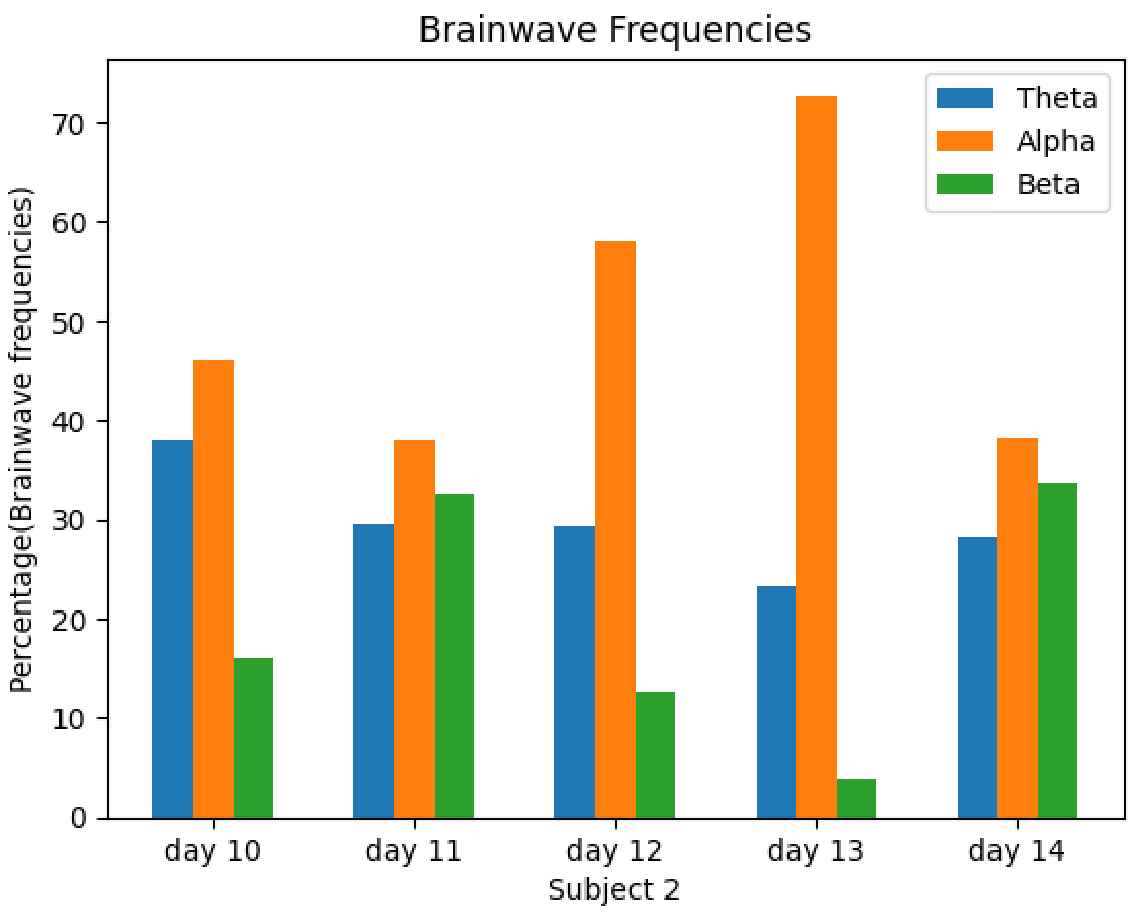

Figure 4: Percentage of brainwave frequencies in **unfocussed state** of Subject 2 using all channels

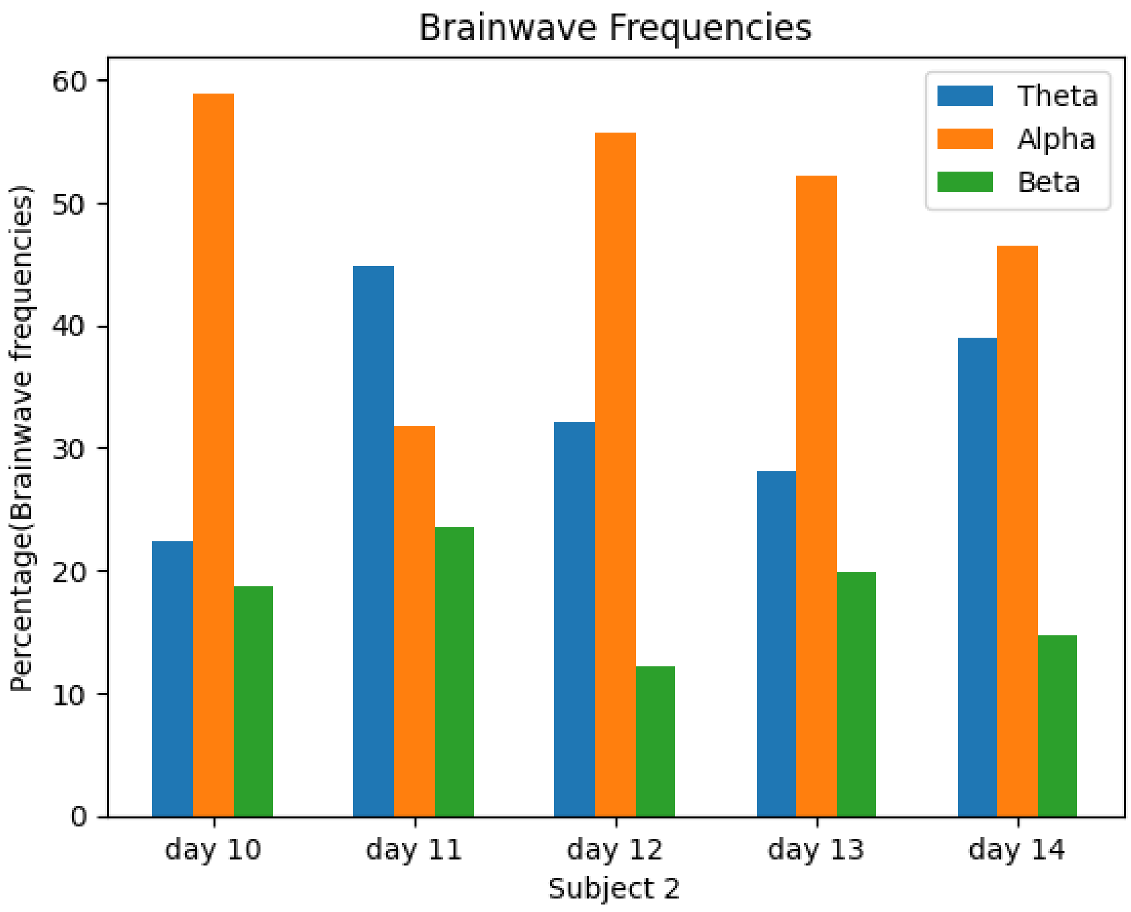

Figure 5: Percentage of brainwave frequencies in **drowsy state** of Subject 2 using all channels

From the above 3 images which correspond to different states of the same subject using all channels we infer that:Drowsy state is marked by high presence of theta and alpha waves. While unfocus state show a mixture of all but slightly increased beta waves than drowsy state.while the focus state shows the highest presence of beta waves.

6.3 Track 3:Brainwaves and their dominant locations in brain.

6.3.1 alpha waves

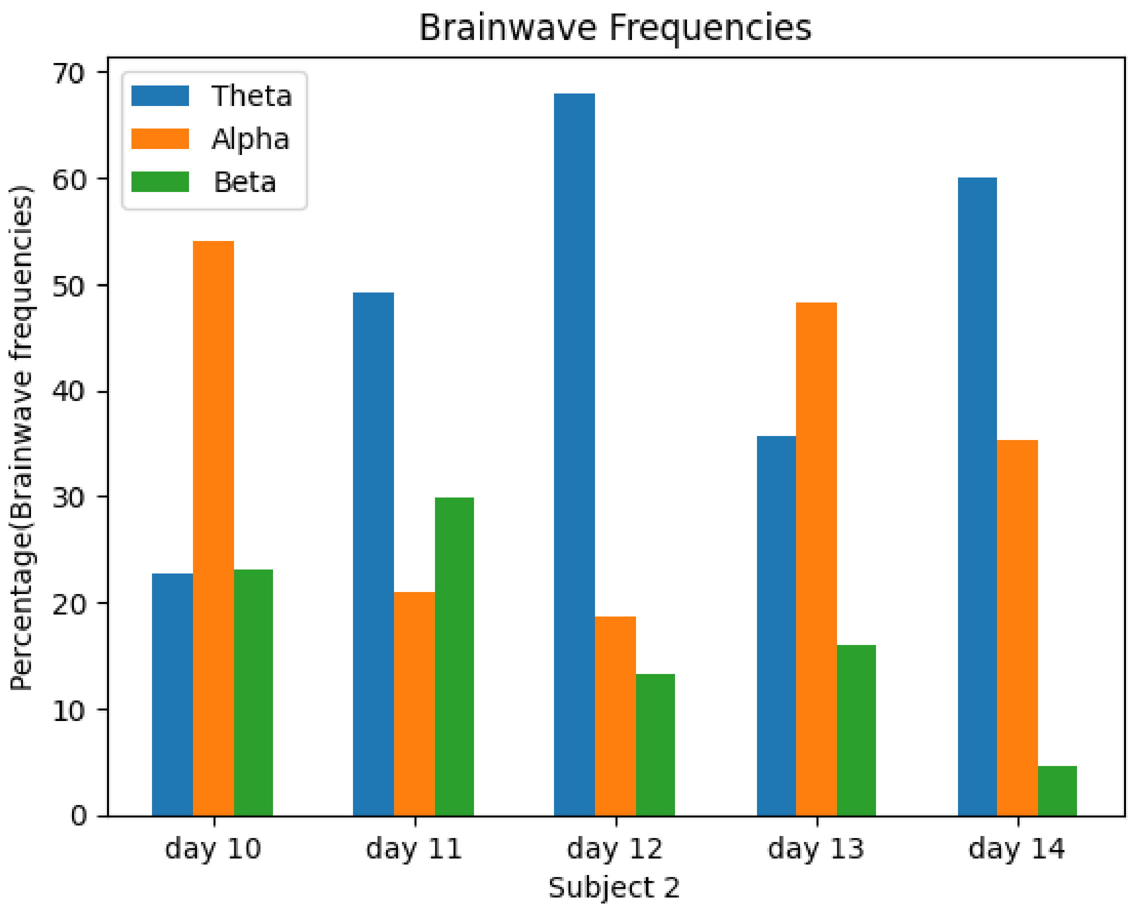

Figure 6: Percentage of brainwave frequencies in drowsy state of Subject 2 using **frontal(F7,F3,AF4) channels**

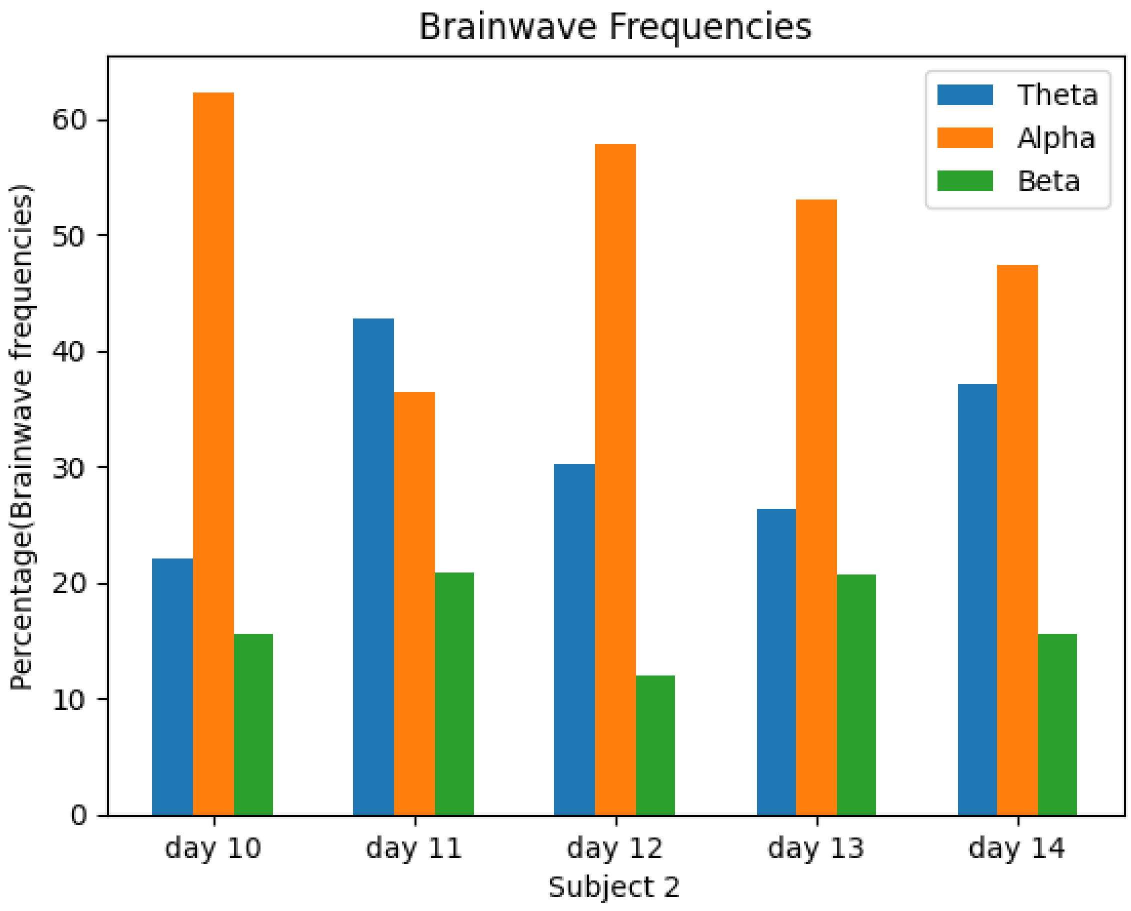

Figure 7: Percentage of brainwave frequencies in drowsy state of Subject 2 using **pareital and occipital(P7,P8,O1,O2) channels**

The given two plot are for drowsy states of subject 2.First plot uses only Frontal electrodes while the second one uses Pareital and occipital electrodes.We can infer that the presence of the alpha waves(orange) is significantly high in the parietal and occipital lobes of brain.

6.3.2 beta waves

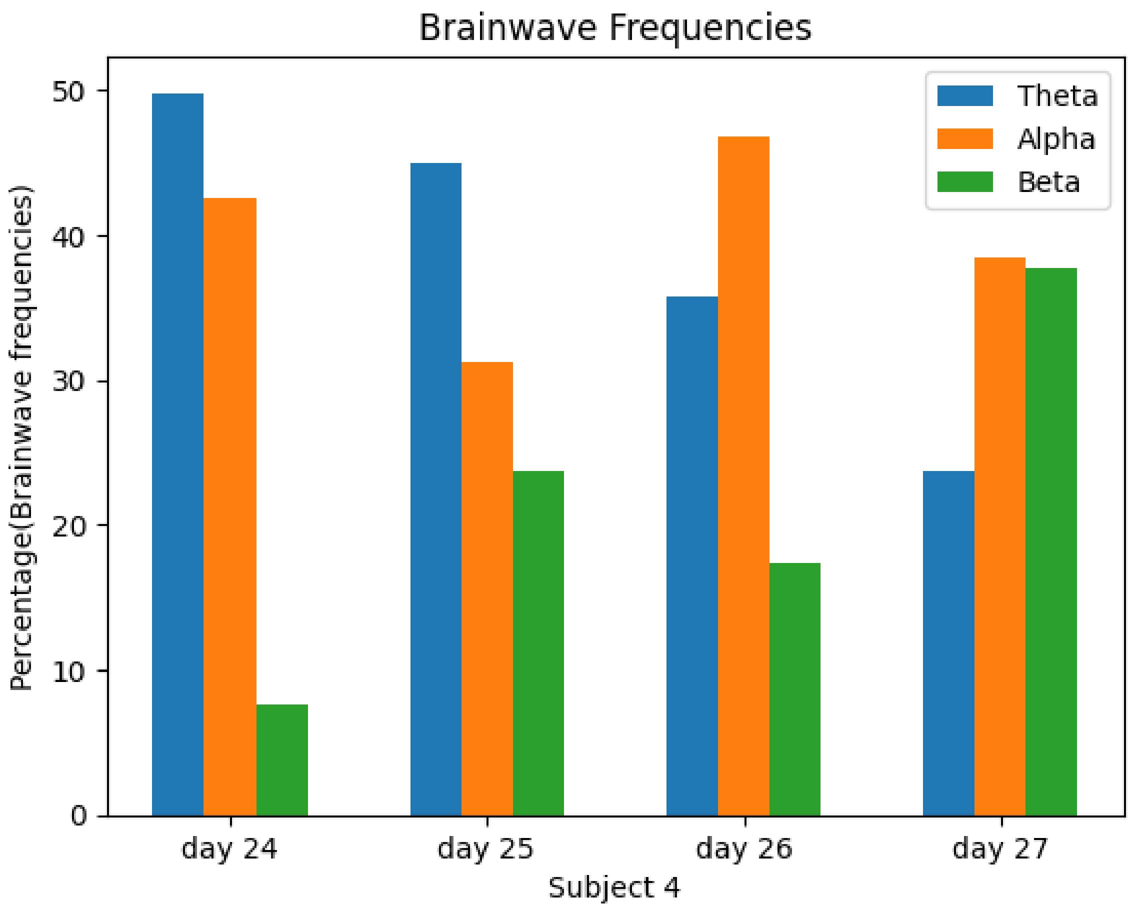

Figure 8: Percentage of brainwave frequencies in unfocus state of Subject 4 using **frontal(F7,F3,AF4) channels**

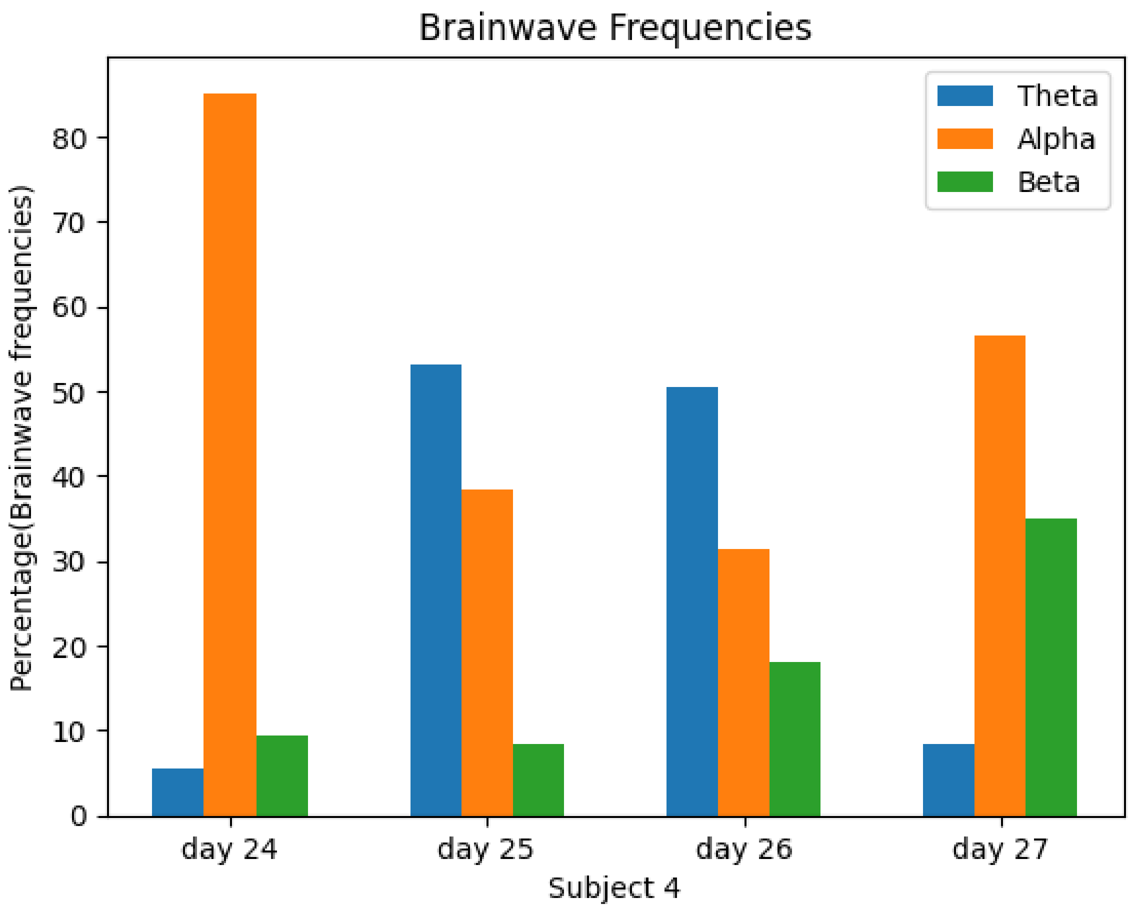

Figure 9: Percentage of brainwave frequencies in unfocus state of Subject 4 using **pareital and occipital(P7,P8,O1,O2) channels**

The given two plot are for unfocus states of subject 4.First plot uses only Frontal electrodes while the second one uses Pareital and occipital electrodes.We can infer that the presence of the low-beta waves(green) is significantly high in the frontal lobes of brain.

6.4 Track 4:Emotional valence states prediction

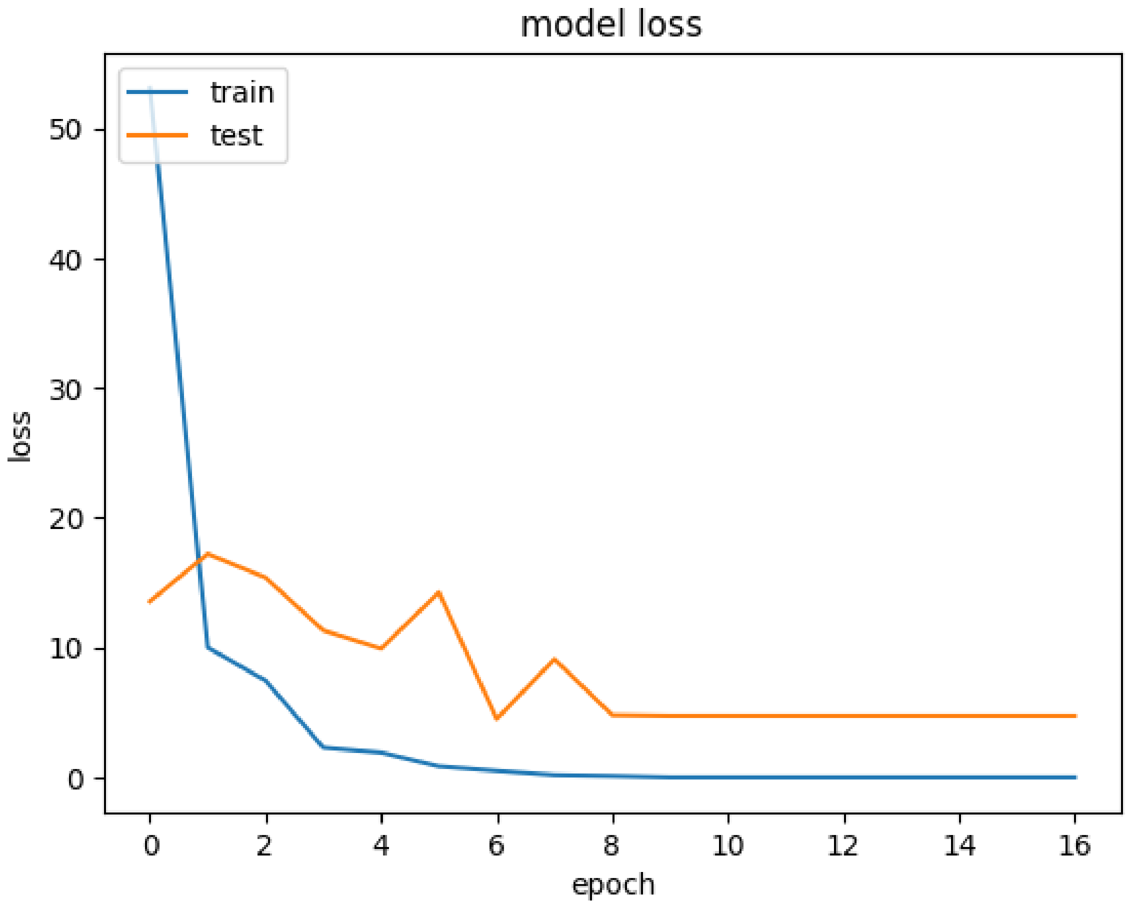

Figure 10: Model loss vs epochs

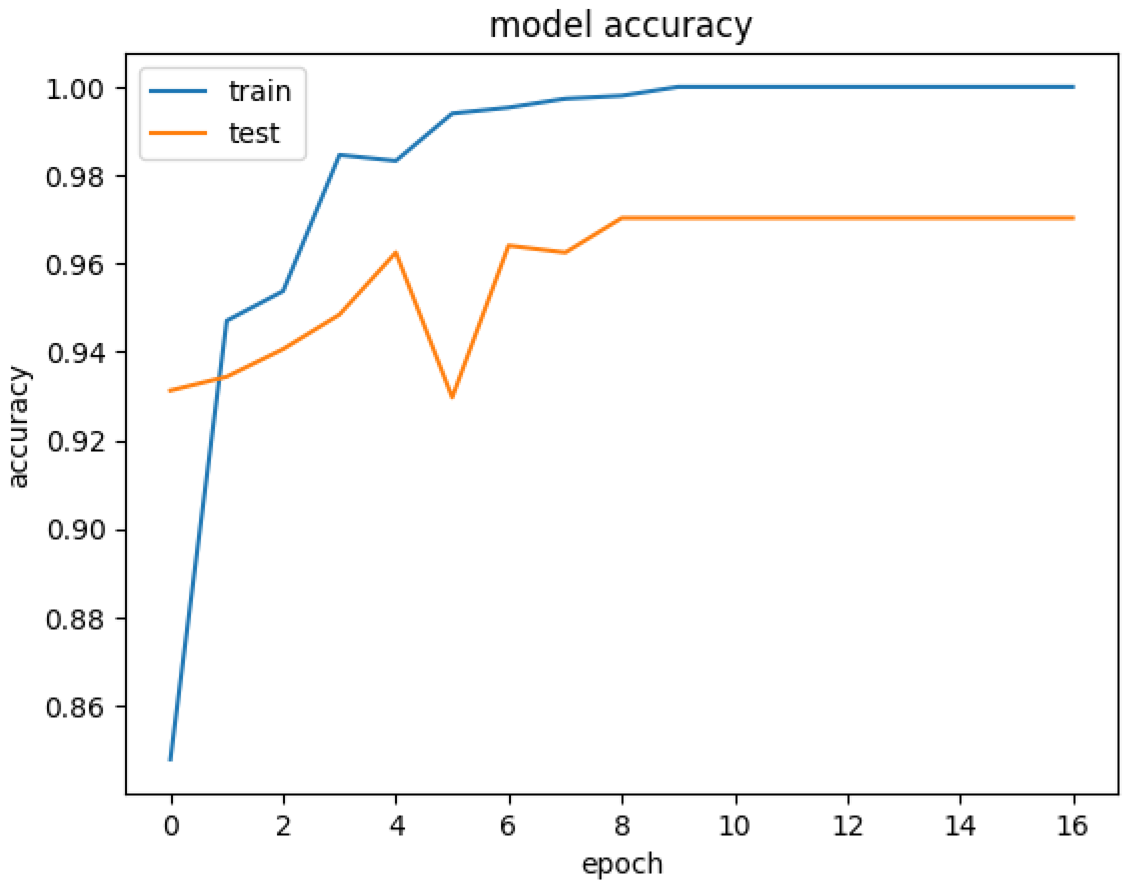

Figure 11: Model accuracy vs epochs

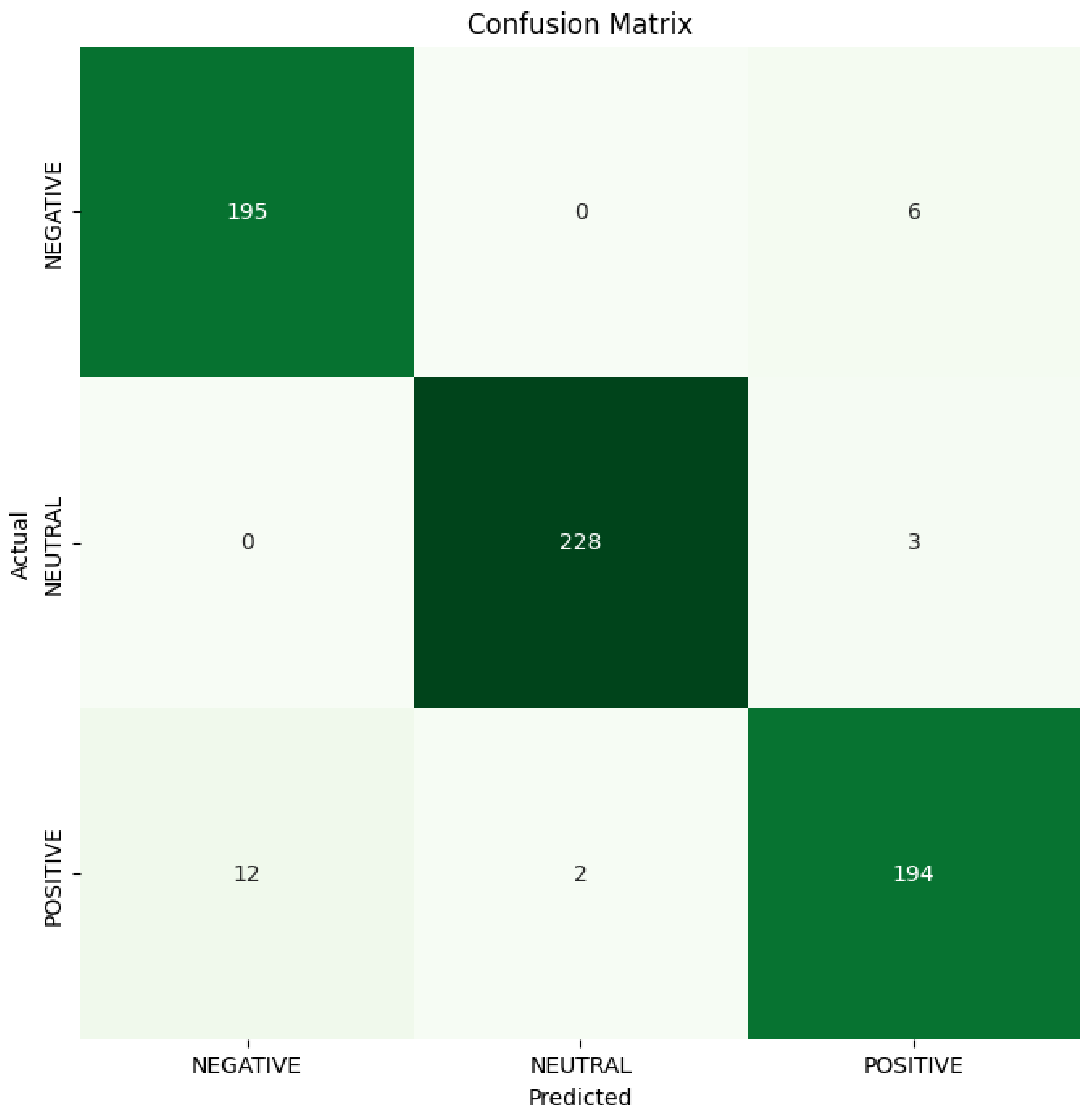

Figure 12: Confusion matrix describing the positive ,negative ,neutral states as predicted and actual markers

7 References

[1] Distinguishing mental attention states of humans via an EEG- based passive BCI using machine learning methods.BY- Çigdem Inan Acı, Murat Kayaa , Yuriy Mishchenko

[2] Prediction of Cognitive Load from Electroencephalography Signals Using Long Short-Term Memory Network .BY-Gilsang Yoo, Hyeoncheol Kim and Sungdae Hong

[3] Mental arithmetic task load recognition using EEG signal and Bayesian optimized K-nearest neighbor Lakhan Dev Sharma1 • Himanshu Chhabra2 • Urvashi Chauhan3 • Ritesh Kumar Saraswat2 • Ramesh Kumar Sunkaria

[4] Mental Emotional Sentiment Classification with an EEGbased Brain-Machine Interface ,Jordan J. Bird,Anikó Ekárt,Christopher D. Buckingham,Diego R. Faria.

[5] Human emotion recognition from EEG-based brain–computer interface using machine learning: a comprehensive review Essam H. Houssein, Asmaa Hammad Abdelmgeid A. Ali Neural Computing and Applications