Group Project CSEE CE-903

Title: Machine Learning to Classify Motor Imaginary Left and Right-Hand Movements

(Team: 11)

Software Requirements Specification Document

Team members

RAMIREZ STANFORD JAIME	: 2100952
RAY DISA	: 2100850
RAZA AHMAD	: 2101194
RAZA ALI	: 2101125
REHMAN IBAD UR	: 2107594
SAELEE THAKDANAI	: 2109371
THIRUMALAREDDY SUNDARI	: 2101101
VALADEZ MENA RIGOBERTO	: 2100374

Submitted to

Supervisor:	Jichun Li
Second Assessor:	Zilong Liu
Start:	02.10.2021
Final Submission:	22.03.2022

Abstract

The goal of this project is to create a motor-imagery classification system using Brain Signals. It is aimed at implementing filtering, feature selection, feature extraction on the signals. The pre-processed data is then fed into Machine Learning/Deep Learning models for training which is then used for predicting unlabeled data. The trained model could then be utilized for controlling a robot arm which is a highly useful device for people suffering from paralysis or severe neuro-muscular diseases.

A Graphical User Interface has also been implemented which allows users to view the Models achieving highest accuracy and the accuracy achieved when we use the model for prediction. Furthermore, this paper also describes the methods in which the Project had been Managed and a description on the techniques we implemented and the results that were achieved.

Keywords – *Electroencephalography (EEG) signals, Signal Filtering, Feature Selection, Feature Extraction, Machine Learning, Deep Learning, Graphical User Interface*

Contents

1. Introduction

People with severe neuromuscular disorders, such as late-stage amyotrophic sclerosis (ALS) and those paralyzed from higher level spinal cord injury are unable to actuate any of their muscles. Communication with the outside world is therefore problematic for the suffering people. Cognitive and sensory body functions, however, are often only minimally affected. Therefore, an electroencephalogram (EEG)-based communication which does not require any neuromuscular control is particularly helpful to enhance the disabled's quality of life by promoting their independence [1].
Besides EEG, there are other techniques for monitoring brain activity, such as functional magnetic resonance imaging (fMRI), magnetoencephalography (MEG), positron emission tomography (PET) or single photon emission computer tomography (SPECT). The advantages of those methods are better accuracy and better spatial resolution compared to EEG. However, due to their large size, heavy weight and high price, they are not as suitable for BCI applications as EEG. Furthermore, EEG offers a better temporal resolution (recording the active state of the brain), portability and relatively low cost. Recently, low-cost consumer devices came on the market (e.g., EMOTIV EPOC+), which will further push the advancements in the area of EEG-based BCI [2].
In general, the information flow in a BCI follows the following path: signal acquisition, signal (pre)-processing, feature selection and extraction, classification (detection of distinct signal patterns), application interface (e.g., to a robot arm), and feedback as displayed in Image 1.1.

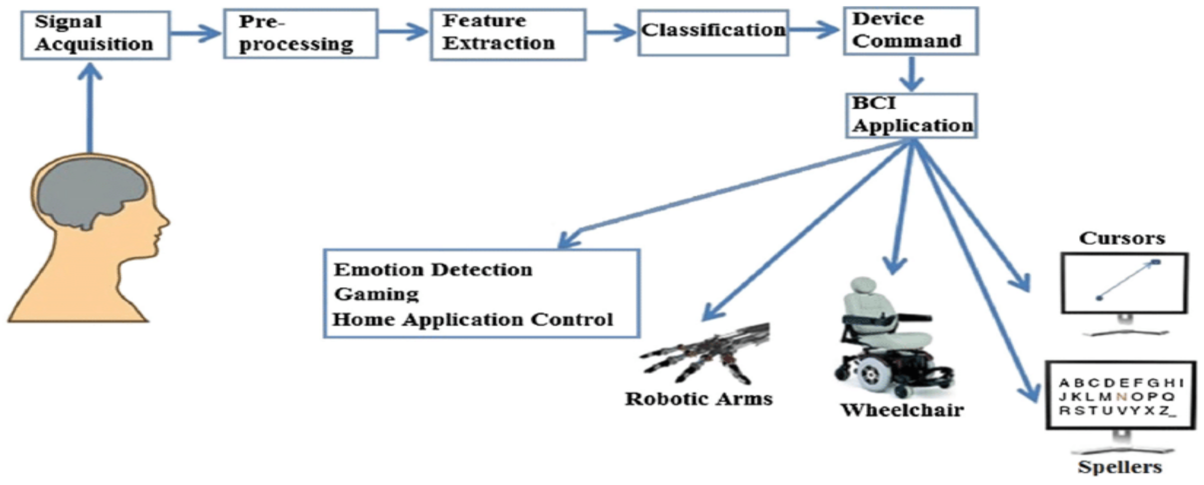


Image 1.1: Information flow for a BCI system controlling a robotic arm [Image from 23].

The human brain has several regions/cortexes which deal with different nerves in the human body. For our purpose, we are mostly concerned with the motor cortex (Image

1.2) which is responsible for imaginary right/left hand movements. To record data/activity in the motor-cortex the electrodes ‘C3’, ‘Cz’ and ‘C4’ are used and are place in the left, center and right of the cerebral cortex respectively. Each electrode signal can also be referred to as different EEG channels and can be treated independently in the process of signal pre-processing.

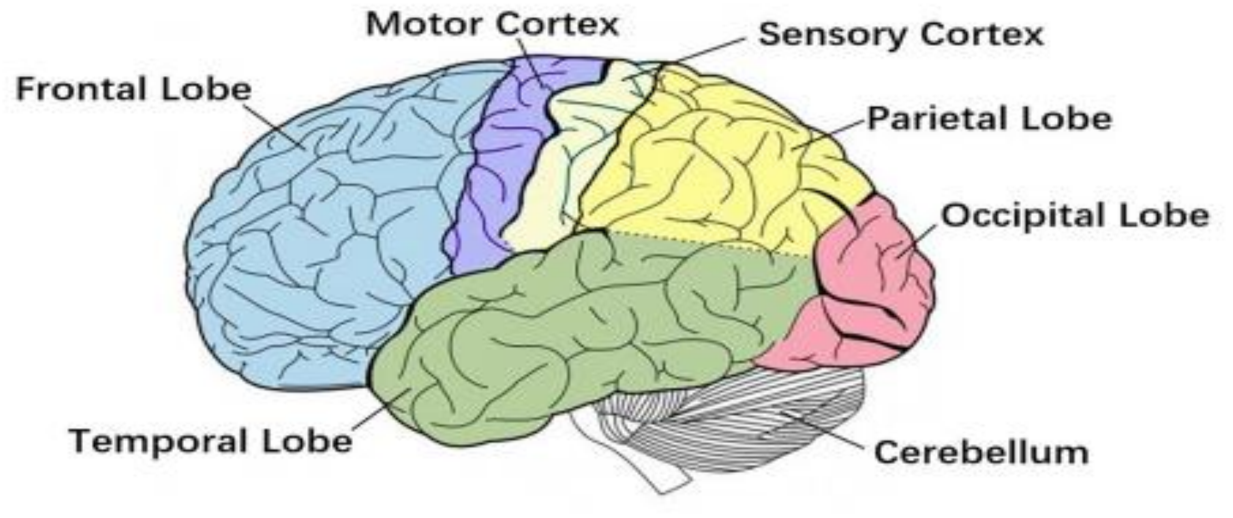


Image 1.2: Brain Cortexes [Image from 24]

The main steps involved in processing BCI data include Signal pre-processing, Feature Selection and Feature Extraction. BCI data which is recorded on subjects involve noise which occurs when activities such as blinking of eyes, daydreaming, dozing off, loss of concentration, itch, etc. occurs. Since we are only concerned with the part of the signal which involves imagining hand-movements it is necessary to eliminate the slow drifts in data. For this purpose, we apply filtering techniques such as Notch Filters, Band-Pass Filters, High-Pass filters etc.
Furthermore, it can be useful to understand the various Brain Waves before applying filtering to understand concentrating on which waves/frequencies serve our purpose. As displayed in Image 1.3, Brain Waves are of 5 types – Delta (0.5 – 4 Hz), Theta (4 – 8 Hz), Alpha (8 – 13 Hz), Beta (13 – 32 Hz) and Gamma (32 – 100 Hz). Delta Waves are more prominent while sleeping, Theta Waves are more evident during Day-Dreaming/Drowsiness, Alpha Waves are for the restful state of mind, Beta Waves are present when the subject’s min is thinking/active and Gamma Waves correspond to the Concentrated state of mind.

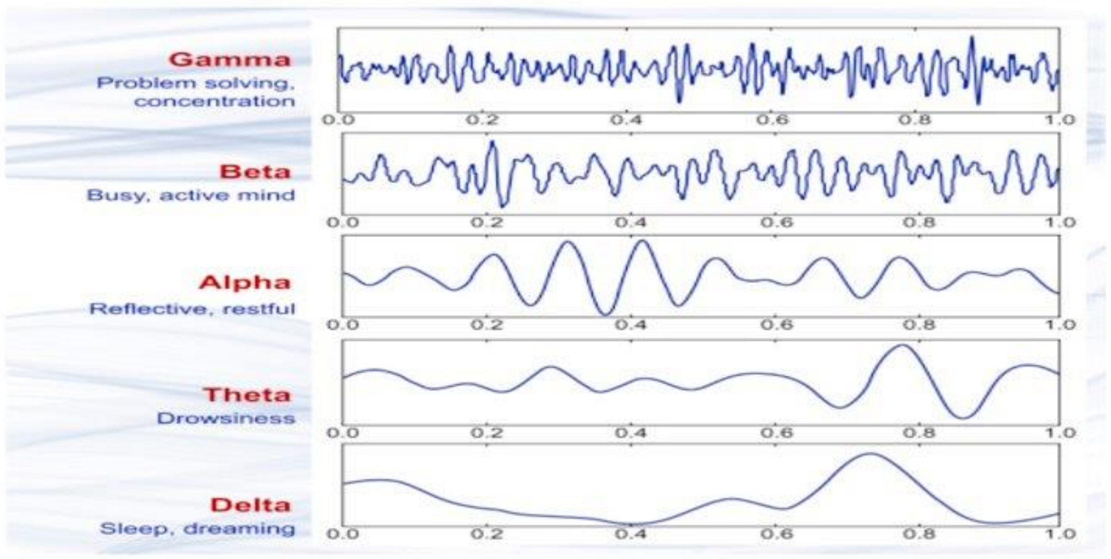


Image 1.3: Brain Cortexes [Image from 25]

Alternatively, Feature Selection can also be performed to segregate artifacts and remove noise. Some Feature Selection techniques include Independent Component Analysis (ICA), Principal Component Analysis (PCA), etc. Consequently, Feature Extraction is used to concentrate only on the important characteristics of the signal, which can be useful while training in Classification models. There are various feature extraction techniques which exist in time domain, frequency domain or the time-frequency domain. Few significant Feature Extraction techniques include Wavelet Transforms, Common Spatial Patterns (CSP), etc.

After performing these steps, the processed signal data is fed into Machine Learning models for training. We get different results for different models as each model works differently and provide different performances depending on the data.

In this paper, we will elaborate our implementation techniques, discuss how our project was managed and tested and discuss the results we achieved.

2. System Design

2.1 Software and Hardware specifications

2.1.1 Software requirements:

Software Used	Description
Operating System	The product is compatible with both Windows and Linux.
Python IDE	The product utilizes Jupyter notebook and any other Python IDE (PyCharm, IDLE or Spyder).
GUI Interface	The product uses React to implement the GUI interface.
Browser	Any browser which supports CGI, HTML5 and JavaScript can be used.

Table 2.1: Software Requirements (Table from [26])

2.1.2 Hardware requirements:

- This product involves the processing of huge data and hence, we would be requiring a system with a RAM of 8GB or higher.
- A multicore processor – i5 or higher would be required for smooth performance.
- An SSD having at least 256 GB of storage available.[26]

2.2 System architecture

The system is built from the data in BCI competition 2003. After analysis and pre-processing, the data, the model is trained and saved. As per [26], users would interact with the system through the interface platform that allows the user to upload their data and select models for classification as shown in Image 2.1.

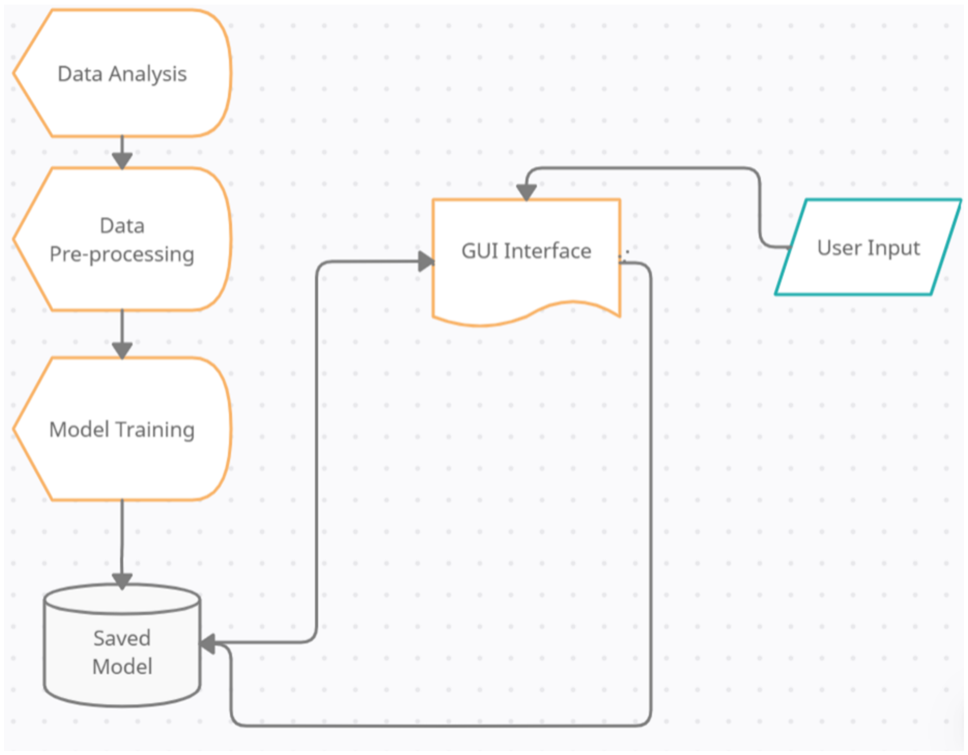


Image 2.1: System Architecture (Image from [26])

Given some browser restrictions, we had to replace the original system architecture and implement a backend. This backend is responsible for loading the models used for predictions and processing the sent data. This has been displayed in Image 2.2.

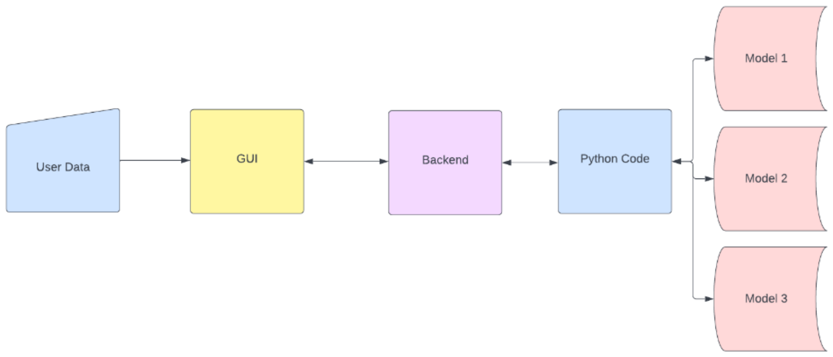


Image 2.2: Revised System Architecture

2.3 Specific requirements

- The system must have at least 60% of accuracy to classify EEG signal.
- A minimum of two traditional machine learning and one deep learning must be applied to the system and verify their performance.
- Implementation of a software GUI platform to interact with the user.

2.3.1 External interfaces:

The input of the system is the signal from electroencephalography (EEG) 3 channel in csv format and the output is the classification of motor imaginary left and right hand movements. The screen format is designed to choose the model for classification and save the output as a file [26].

2.3.2 Functional requirements:

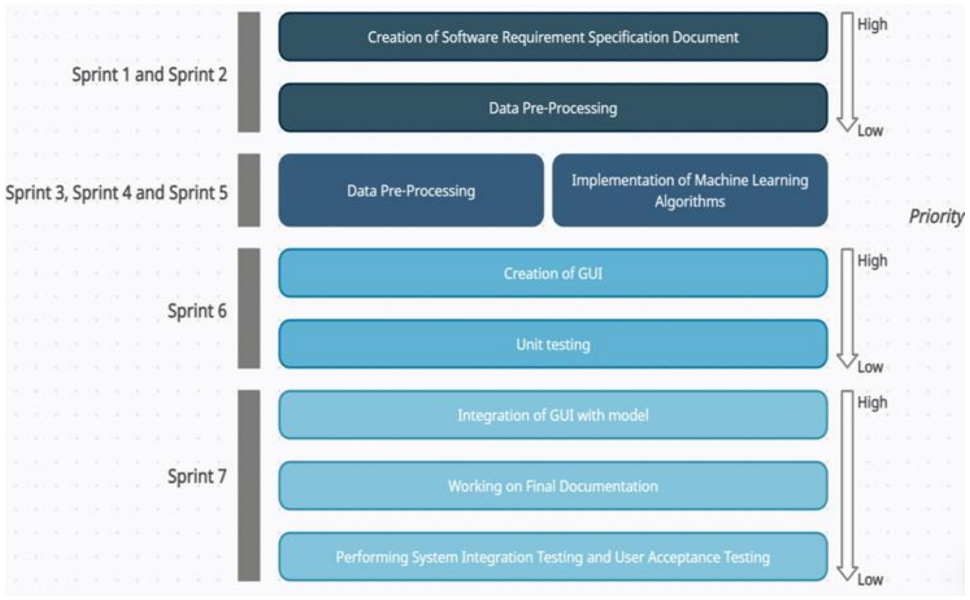

Sprint 1 and Sprint 2
Creation of Software Requirement Specification Document
Data Pre-Processing
High
Low
Sprint 3, Sprint 4 and Sprint 5
Data Pre-Processing
Implementation of Machine Learning Algorithms
Priority
Sprint 6
Creation of GUI
Unit testing
High
Low
Sprint 7
Integration of GUI with model
Working on Final Documentation
Performing System Integration Testing and User Acceptance Testing
High
Low

Image 2.3: Product Backlog (Image from [26])

Here, Data Pre-Processing involves Feature Pre-Processing, Feature Selection and Feature Extraction of Signal Data.

2.3.3 Non-functional requirements

2.3.3.1 Availability

The system is available 24 hours a day, all days of the week [26].

2.3.3.2 Accessibility

The system is accessible to all as there is no level of accessibility present. Every individual having the link to the GUI would is able to get the result displayed on the screen [26].

2.3.3.3 Accuracy

Only the top 3 Machine Learning Algorithms giving an accuracy of more than 65% have been made available to the user through the Graphical User Interface (GUI). These include XGBoost model, ADABoost model and Random Forest Classifier.

2.3.3.5 Testability

The system consists of 3 main units – Data Pre-Processing unit, Prediction Model unit and GUI unit. Each of these units were tested and fixed (in case of any defects) individually (Unit Testing) or entirely (System Integration Testing and User Acceptance Testing) [26].

2.3.3.6 Transparency

To allow users to trust the results, a description of the ML Techniques used are described along with the confusion matrix on the GUI interface for the purpose of transparency.

2.4 GUI Design

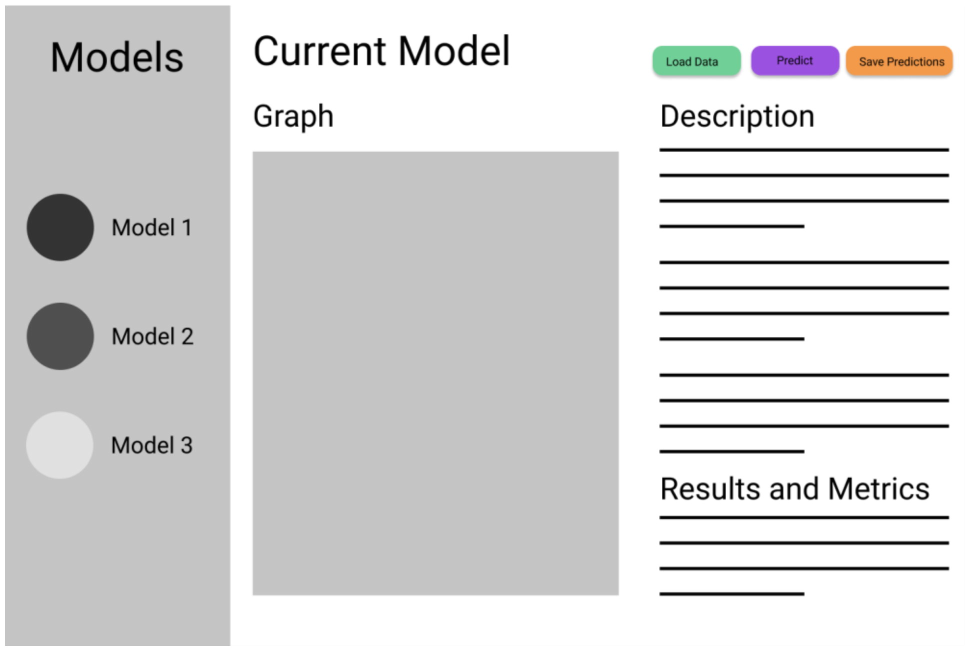


Image 2.4: GUI Design - Wireframe

3. Implementation

3.1 Data Acquisition

To train the classifier model, training data is necessary. The training data used has been obtained by performing trials on a single subject who is 25-year-old female. The subject was asked to relax on a chair which had arm-supports and was told to imagine left- and right-hand movements in order to command a feedback-bar [4].
A total of 280 trials were conducted in 7 runs of 40 trials each. There was a break provided at the end of each run. The total number of trials were randomly split into two halves which consisted of the training set and the testing set. In each trial, the first 2 seconds did not have much activity and after 2 seconds there was a sudden rise in amplitude of the stimulus in the trigger channel indicating some activity as shown in Image 3.1. The main feedback was based on AAR parameters of the channels 'C3' and 'C4' recorded between 3 seconds to 9 seconds [4]. These trials were available in a MATLAB file which consisted of 'x_train', 'y_train' and 'x_test' [5].

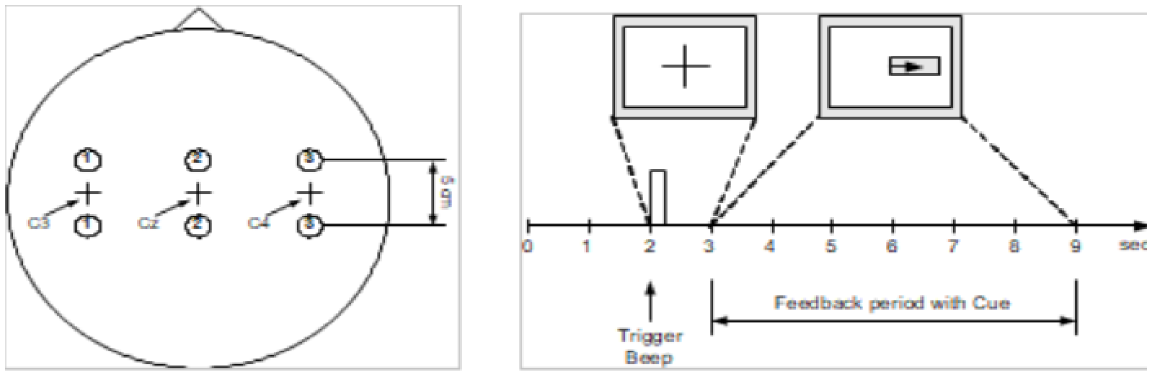
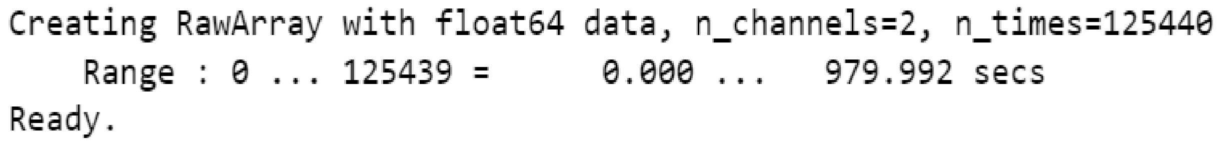


Image 3.1: Position of electrodes of C3, Cz and C4 (left) and timing scheme (right) [4]

For our implementation, we converted the .mat file into .csv files for usage in Python3 coding language. After conversion, it was found that 'x_train' consisted of 140 x 3 x 1152 units of data.
The data was sampled at 128Hz [4] and hence, each second in a trial consisted of 128 units of signal data. We thus discarded the first 256 units of data for every channel as it does not impact the actual decision-making period and contains some noise [4, 5].
Furthermore, we did not include the data from 'Cz' as it does not directly impact the feedback process. Thus, the resulting data consisted of 140 x 2 x 896 units.

3.2 Signal Pre-processing

For ease of signal pre-processing, the data is converted to a Raw-Array format (Image 3.2) using the MNE which is an open-source tool in Python3 used for analysis of EEG data [6].



```
Creating RawArray with float64 data, n_channels=2, n_times=125440
    Range : 0 ... 125439 =      0.000 ...   979.992 secs
Ready.
```

Image 3.2: Raw Array created using MNE Python3

It was then checked if any of the channels were 'bad' or 'noisy' channels by visualizing the data (Image 3.3). However, it concluded that None of the channels were bad or had noise.

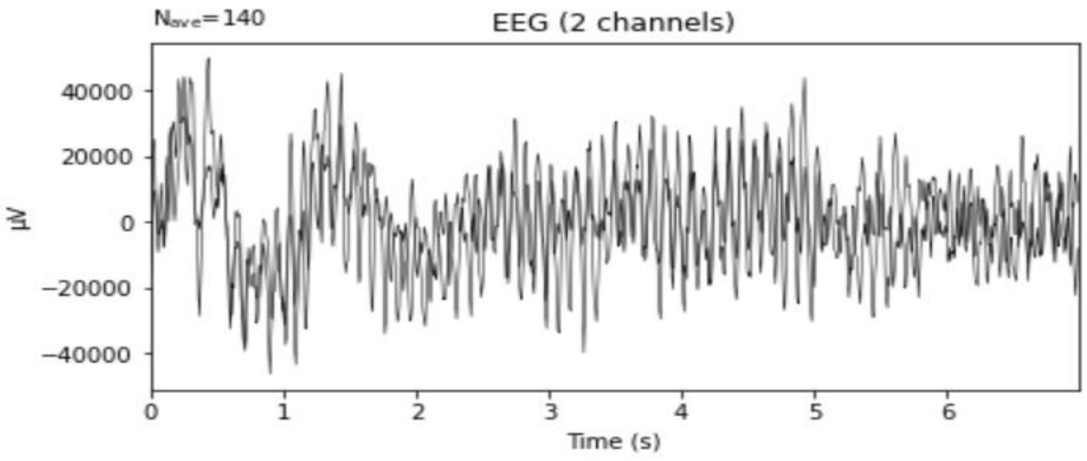


Image 3.3: Visualization of channels 'C3' and 'C4'

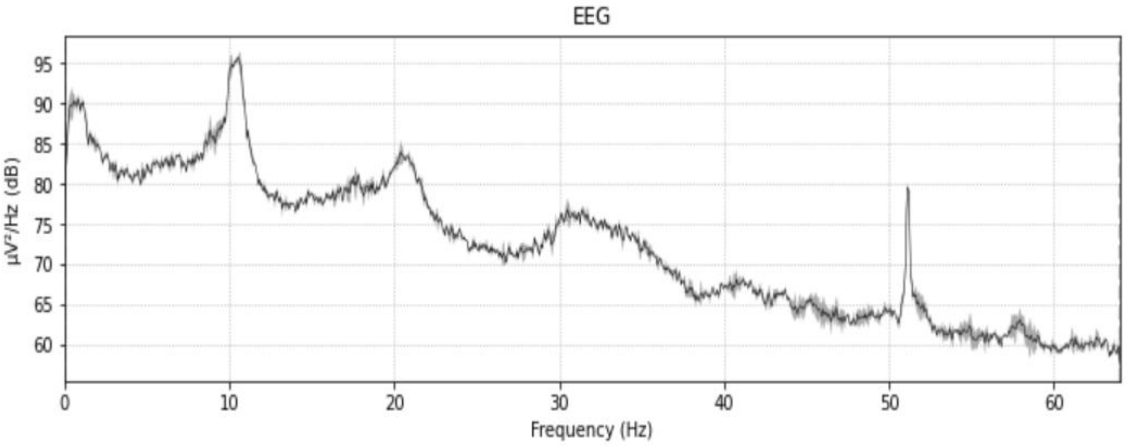


Image 3.4: Visualization of Power Spectral Density of Channels 'C3' and 'C4'

From Image 3.4, it is visible that there exists noise at 10 Hz, 20 Hz and 50 Hz. We could have applied a Notch-Filter to reduce the Power-Line Noise. However, since we would only be focusing Alpha and Beta bands, it was avoided as Notch-Filter only works well for eliminating noise at 50 Hz [7]. Instead, A low-pass filter was applied to filter out frequencies above 40 Hz as shown in Image 3.5.

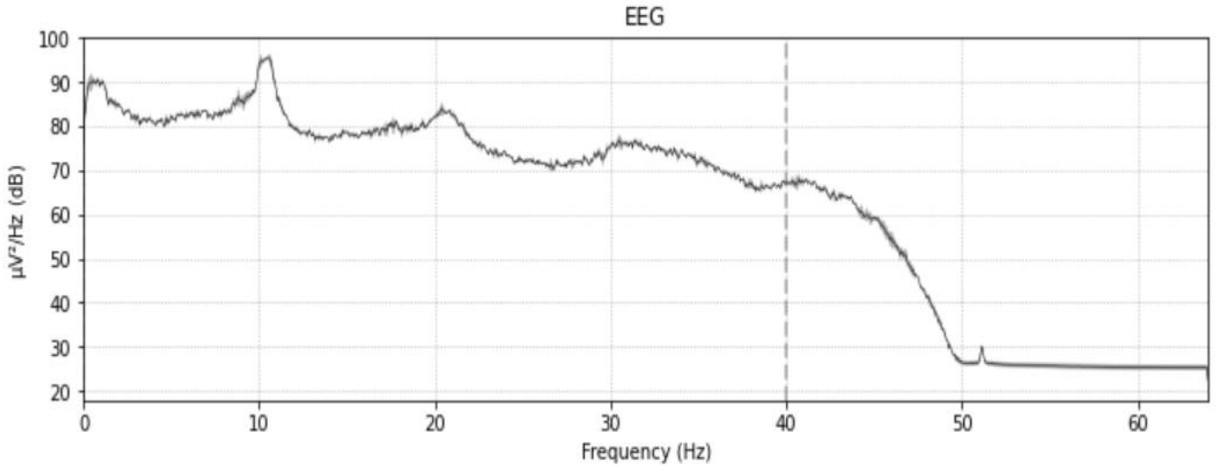


Image 3.5: Application of Low-Pass Filter to filter out frequencies 40Hz and above

A High-Pass Filter was also applied (Image 3.6) which filters out frequencies below 1 Hz. This is done to remove any slow drifts in data which are present in low-band frequencies and might affect further pre-processing [8].

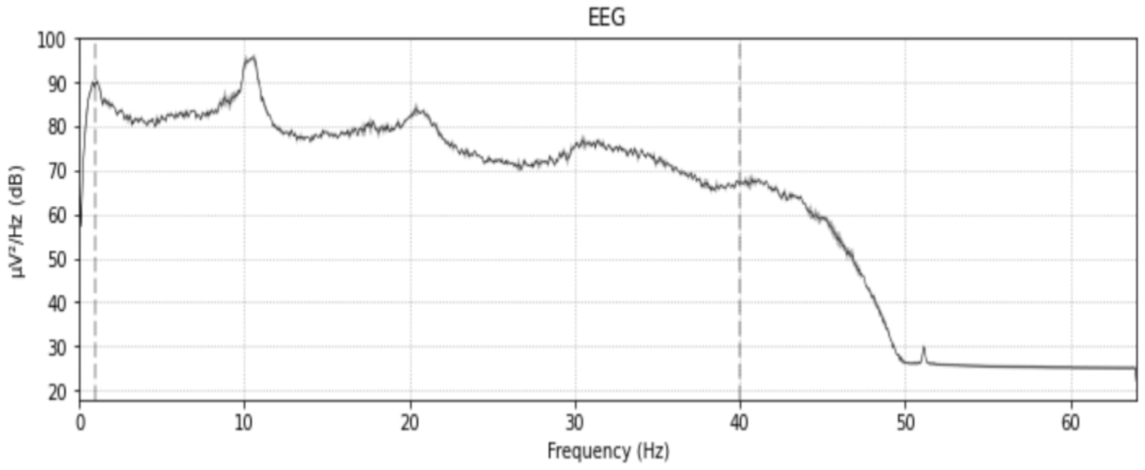


Image 3.6: Application of High-Pass Filter to filter out frequencies below 1 Hz

Independent Component Analysis (ICA) is applied for repairing the signal artifacts. Signals might contain noise components like 'bad blinks', 'jaw clenching', 'swallowing' etc. ICA helps to segragate signals into components in order to eradicate such artifacts [8]. The signals are split into 2 independent components – EEG Channels C3 and C4 and they are analyzed and processed independently.

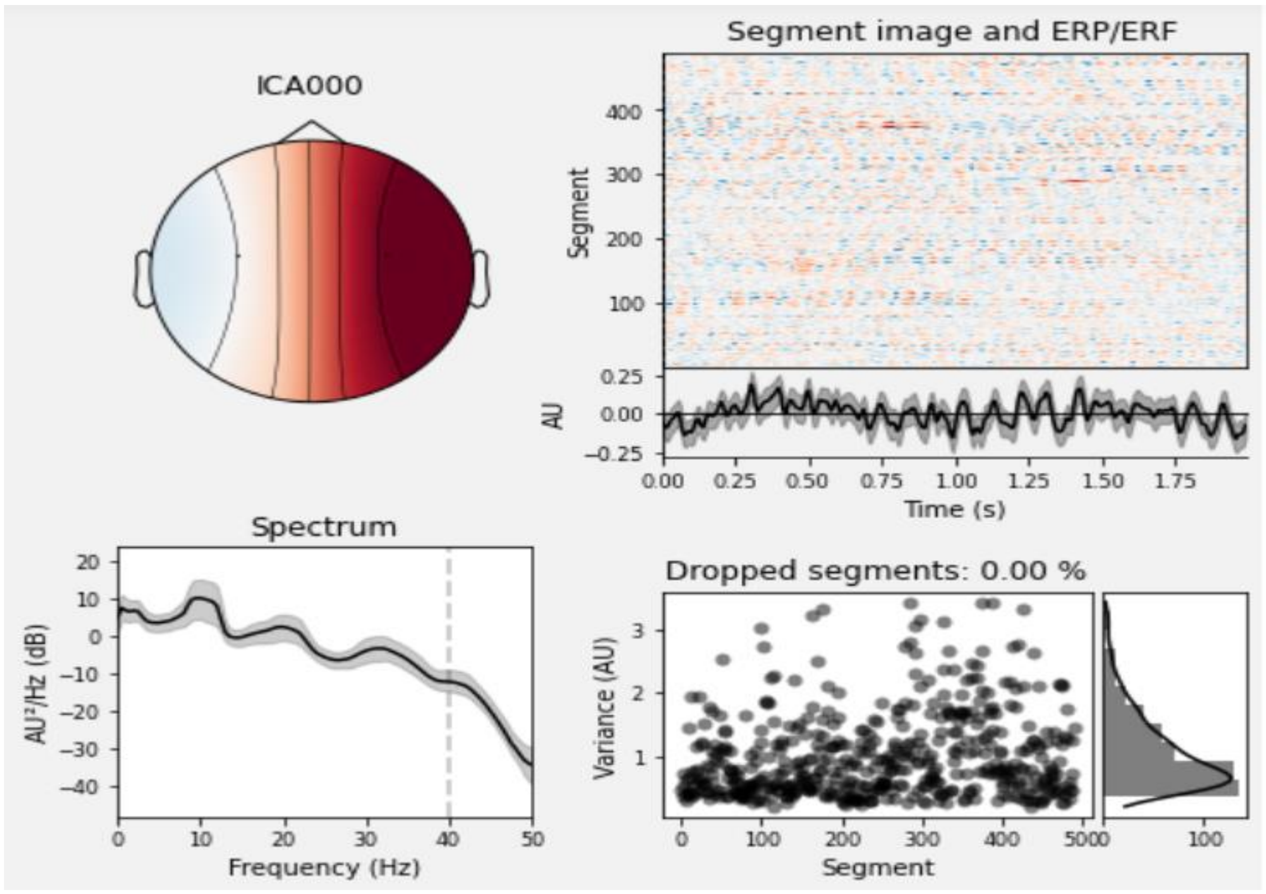


Image 3.7: Independent Component Visualization of Channel C4

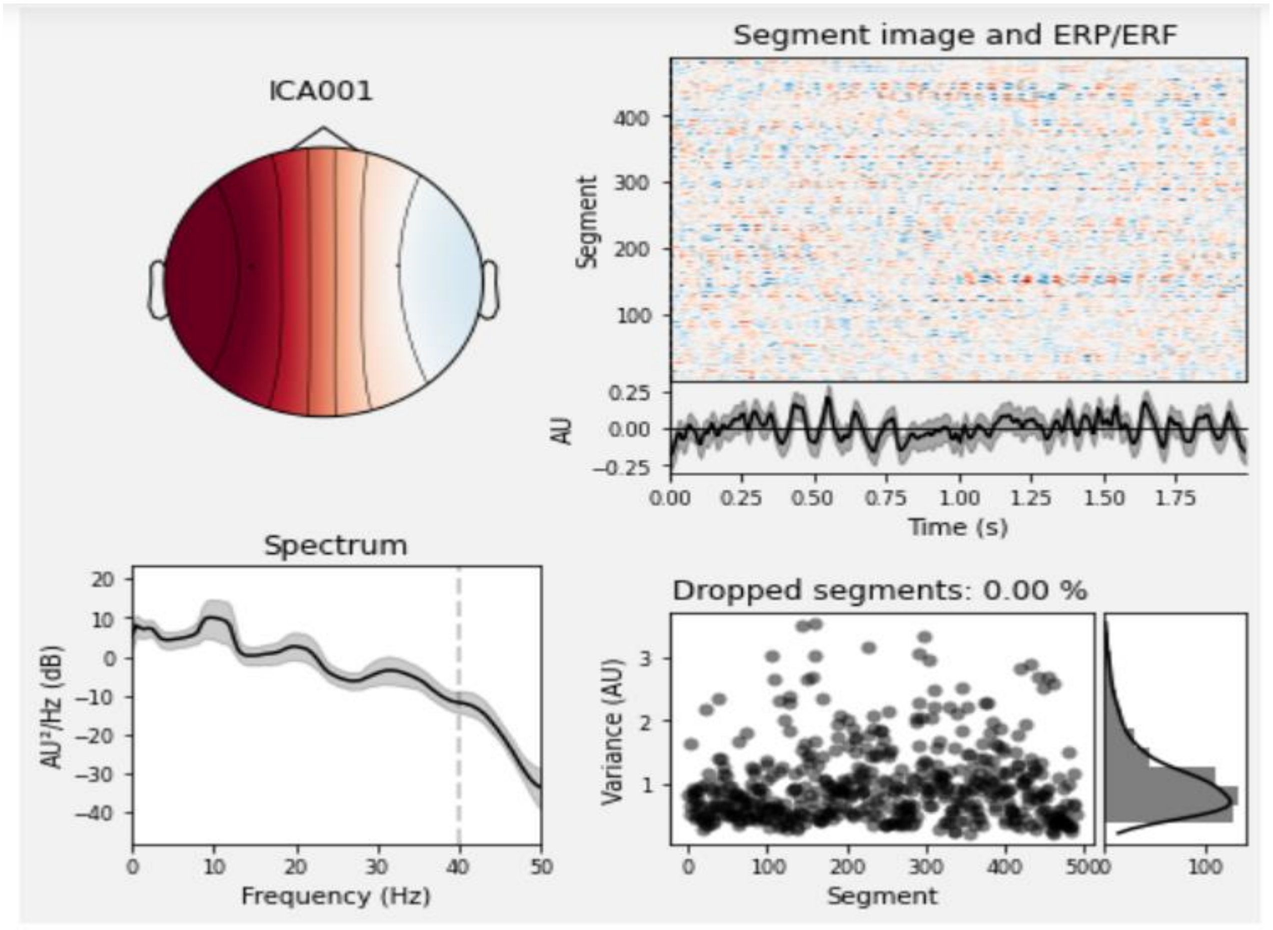


Image 3.8: Independent Component Visualization of Channel C3

From Image 3.7, we can visualize Channel C4 where the power is concentrated on the left hemisphere of the brain which handles the left-hand imagination. Similarly, from Image 3.8 it is evident that in Channel C3 power is concentrated on the right hemisphere of the brain which is responsible for right-hand imagery.

The raw data is converted into Epochs which are then averaged over to check for Event-Related Desynchronization (ERD)/ Event-Related Synchronization (ERS).

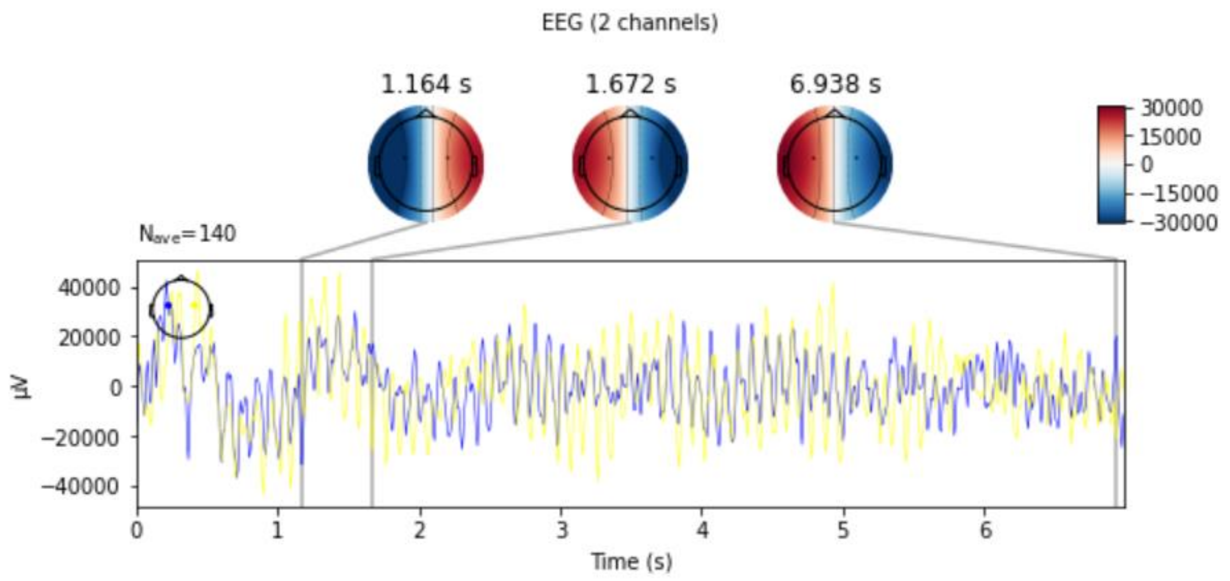


Image 3.9: Averaged Data over Epochs

From Image 3.9, it is evident that ERD/ERS exists throughout the entire time period of 7 seconds. ERD is an amplitude decrease of rhythmic events while ERS is an amplitude increase of the same [9]. ERD is present in the first 3 seconds or the time when the decision period starts, and we mostly observe ERS later throughout the trial.

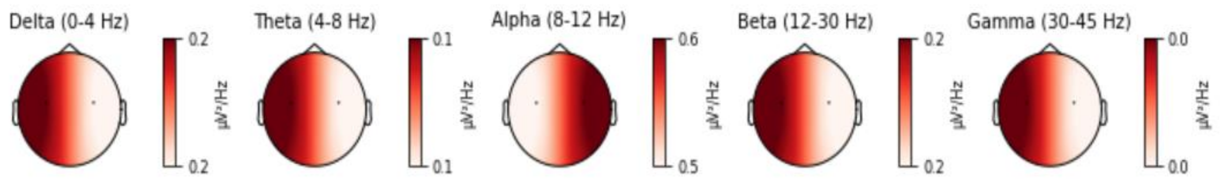


Image 3.10: Power Spectral Density (PSD) of bands Delta – Gamma

The Power Spectral Density (PSD) is visualized among all the frequency bands Delta-Gamma in Image 3.10. PSD depicts the amount of power in a signal throughout all frequency bands. We can understand that the Alpha band has the most power contributing to the decision-making process while the Gamma band (which is used when intense concentration is needed) does not contribute to the decision-making process. This is clear as the subject had been relaxing while the trials were performed.

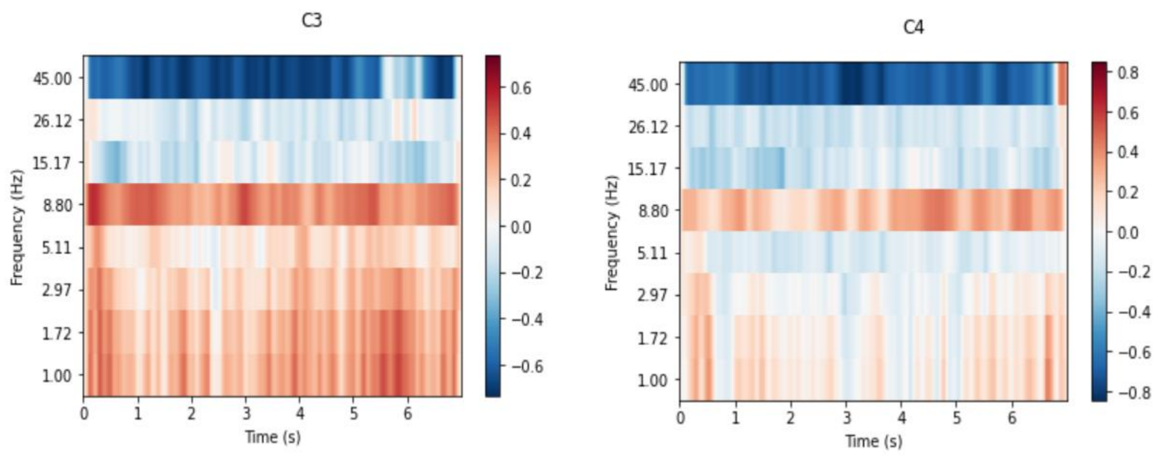


Image 3.11: Inter-Trial Coherence (ITC) over Channels ‘C3’ and ‘C4’

Image 3.11 depicts the Inter-Trial Coherence (ITC) in Channels ‘C3’ and ‘C4’. ITC is a measure of Coherence of phases across trials [10]. We can visualize that there are high positive oscillations in the theta and alpha bands where there is high negative oscillation in the gamma band.

3.3 Feature Extraction/ Feature Engineering

After processing the data and converting the epoch data to a data frame, Feature Extraction has been applied. In order to achieve the highest accuracy and performance in the models at a later stage, good feature extraction was necessary. Therefore, research was done about common feature extraction methods and features that are frequently used in EEG signals. There are many interesting features that we can apply for the model, for example, Maximum linear cross-correlation, Correlation Coefficients in a time domain and power spectrum as well as Standard Features including Mean, Standard deviation, Peak-to-Peak Amplitude, Root-Mean Squared Value, Quantile, etc. The best features are identified by looping through a combination of features and finding the best set of features giving a good prediction score.

There are various methods that have been implemented for Feature Extraction:

1. Common Spatial Pattern (CSP)

CSP is considered to be the most effective Feature Extraction Technique which could be applied to Two-Class Classification problems [11], [12] and [13]. It allows us to extract spatial features of ERD/ERS in the signal, thus increasing the power difference between the two classes that we are predicting [14]. This allows us to have better classification accuracy.

2. MNE Feature Extractor [18]

We have also utilized the MNE Feature Extraction Tool to select features that would improve our Machine Learning Accuracy. This has been done in an iterative manner to select the combination of features that work best as shown in Image 3.12. We tried iterating through the following list of features:

Correlation Coefficients (Time Domain): This feature helps in identifying the time delay between two signals and is a very useful feature to identify the linear relationship between two-variables [15].

Approximate Entropy: This is a measure which roughly quantifies the uncertainty in EEG data [16].

Quantile: This measure determines the number of values in a distribution above and below a certain limit [17]. The limit can be any floating number between 0 and 1.

Mean: This variable determines the average value of data over a range.

Standard deviation: This variable determines the amount of variation or dispersion of a set of values.

Variance (per channel): This variable determines how far a set of numbers from their average value.

Peak-to-peak amplitude: This variable measure between highest amplitude value and lowest amplitude value.

Root mean square (rms): The square root of the arithmetic mean of the squares of the values.

Quantile: This variable determines cut points dividing the range of a probability distribution (default value is 75) [18].

Maximum linear cross-correlation: Maximum value of two series as a function of the displacement of one relative to the other.

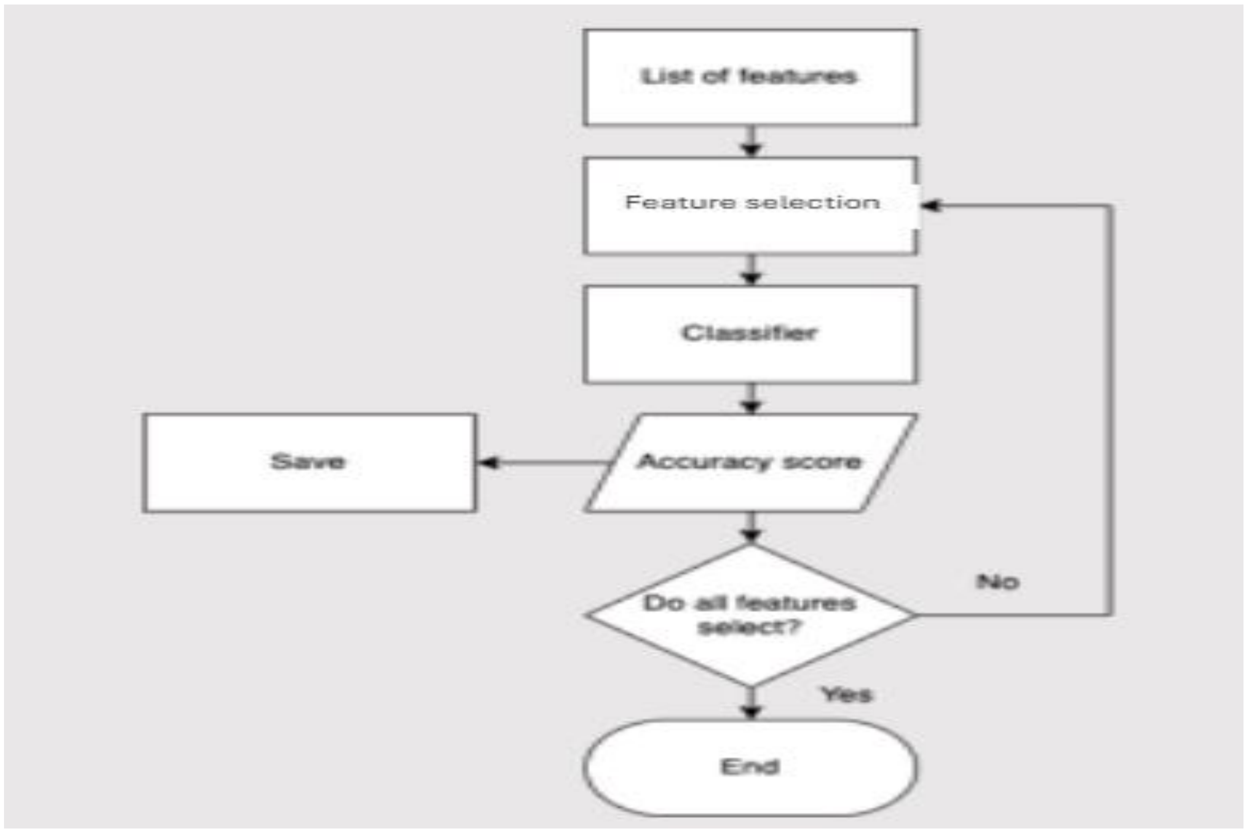
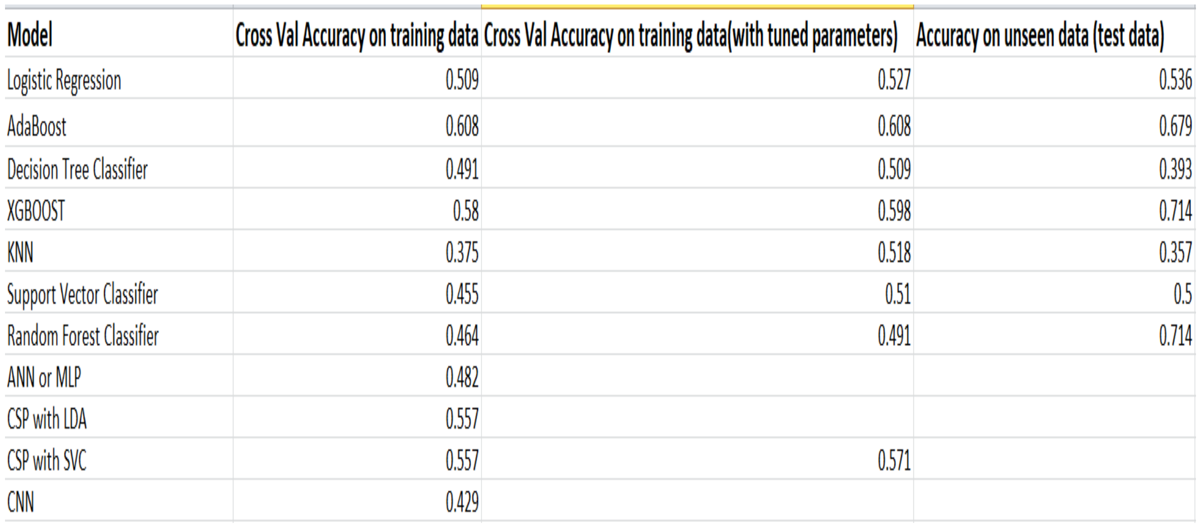


Image 3.12: Feature Extraction – Iterative Method

3.4 Classification

The main goal of the BCI project is to translate recorded brain activity into a robot command. The last step in the control chain involves the identification of feature patterns in order to classify the user's intent. The output of the classification stage is the control input of the robot arm. One of the objectives of the system is to provide multiple classifier options to the user. For this purpose, we tested Eleven classifiers including two deep neural network model as shown in Image 3.13.



Model	Cross Val Accuracy on training data	Cross Val Accuracy on training data(with tuned parameters)	Accuracy on unseen data (test data)
Logistic Regression	0.509	0.527	0.536
AdaBoost	0.608	0.608	0.679
Decision Tree Classifier	0.491	0.509	0.393
XGBOOST	0.58	0.598	0.714
KNN	0.375	0.518	0.357
Support Vector Classifier	0.455	0.51	0.5
Random Forest Classifier	0.464	0.491	0.714
ANN or MLP	0.482		
CSP with LDA	0.557		
CSP with SVC	0.557	0.571	
CNN	0.429		

Image 3.13: Results of various Classifiers

3.4.1 Random Forest

Random Forest is a popular classifier due to its performance and robustness. It is a supervised learning algorithm that essentially builds multiple uncorrelated decision trees to form a forest which produces higher accuracy. To avoid issues like bias and overfitting, Random Forest adds randomness by generating a random subset of features and searching for the best features instead. Random Forest algorithm is used for both the classification and regression problems. It handles overfitting issues quite well as compared to Decision Trees and trains faster but is resource intensive as they compute data for each individual decision tree.
Random Forest has been used extensively in the past for BCI classification applications. While [19] classifying a P300-based Brain-Computer Interface system, Random Forest was compared against Support Vector Machine, stepwise linear discriminant analysis (SWLDA); and the multiple convolutional neural networks (MCNN) and the ensemble support vector machine (ESVM).
In the above study, Random Forest achieved the highest classification accuracy even better than the state-of-the-art ESVM and MCNN.
All the above makes Random Forest a strong candidate for this project where we need to classify BCI signals. There are three hyper-parameters that are tuned before the actual training begins. These parameters are node size, number of trees, and the maximum number of leaves in each decision tree. We used the following parameters for training the model.

Number of trees: 400, 550, and 600

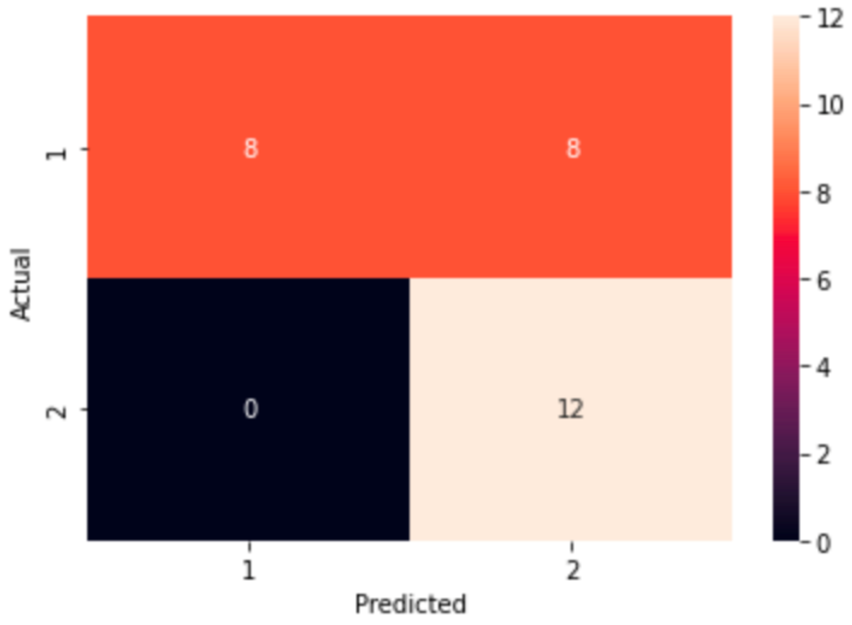
Node size: 3, 9, and 10.

Maximum Depth: 3, 10, and 19.
We further applied cross-validation (K-fold) for better training results and to avoid

overfitting and finally used GridSearch to set the best parameters for our model. The following table shows the difference in accuracy before and after applying the optimized parameters.

Cross Validation Accuracy (Training data)	Cross Validation Accuracy (Training data with tuned parameters)	Accuracy on Unseen Data
51%	47%	75%

Table 3.1 Random Forest cross-validation accuracy



Confusion Matrix - Random Forest

Random Forest cross validation accuracy on training data was 45% which is improved after supplying with optimized parameters. It scored reasonably well on unseen data with the accuracy of 75%.

3.4.2 Support Vector Machine

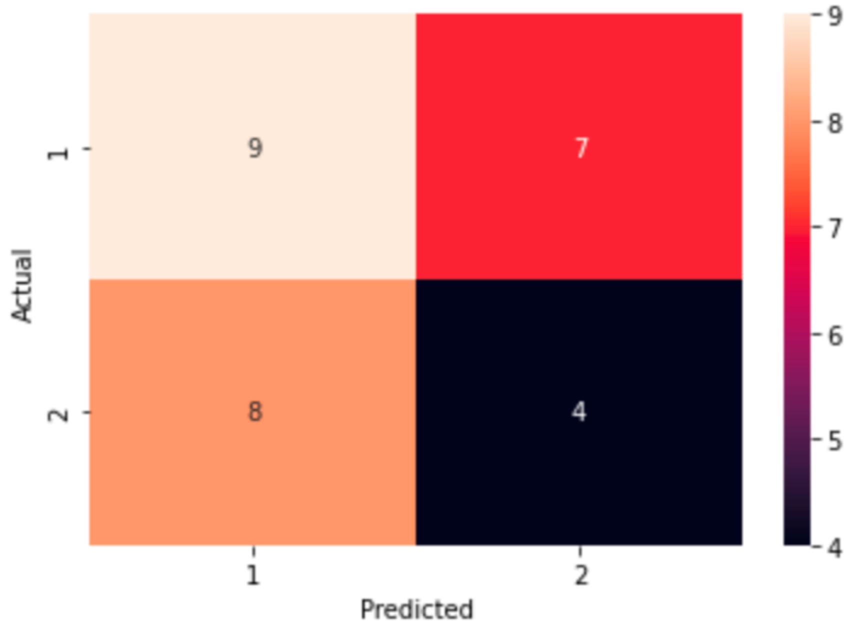
Support Vector Machine (SVM) is another very powerful algorithm for classification and regression problems. SVM produces high accuracy and is computationally less expensive. SVM classifies by finding the hyperplane in an N-dimensional space. It aims to find the hyperplane that contains a maximum number of margins that represents the maximum distance between two different classes. The whole concept of SVM lies on support vectors which are the points close to these planes and can greatly influence the position and shape of the hyperplane.

Support Vector Machine proved highly effective in classifying words in a [20] BCI competition 2003-data set IIb. The algorithm was trained on optimal parameters achieved from a 5-fold cross-validation process to classify unlabelled data and achieved an accuracy of 84.5%. In another study [21] Support Vector Machine performed better than Artificial Neural Networks in a BCI system to control wheelchair movements. SVM scored above 90% average accuracy across ten different testing subjects while Artificial Neural Networks scored not more than 84%.

Given the success of Support Vector Machine in similar BCI system problems, we used SVM with cross-validation and tuned its hyperparameters using GridSearch. We tested against multiple gamma values (learning rate) as well as both radial and linear kernels. The best parameters for our dataset were gamma =1 and kernel was radial. SVM produced the following results when tested with cross-validation and then with tuned parameters.

Cross Validation Accuracy (Training data)	Cross Validation Accuracy (Training data with tuned parameters)	Accuracy on Unseen Data
45%	54%	64%

Table 3.2 SVM cross-validation accuracy



Confusion Matrix – Support Vector Machine

3.4.3 Logistic Regression

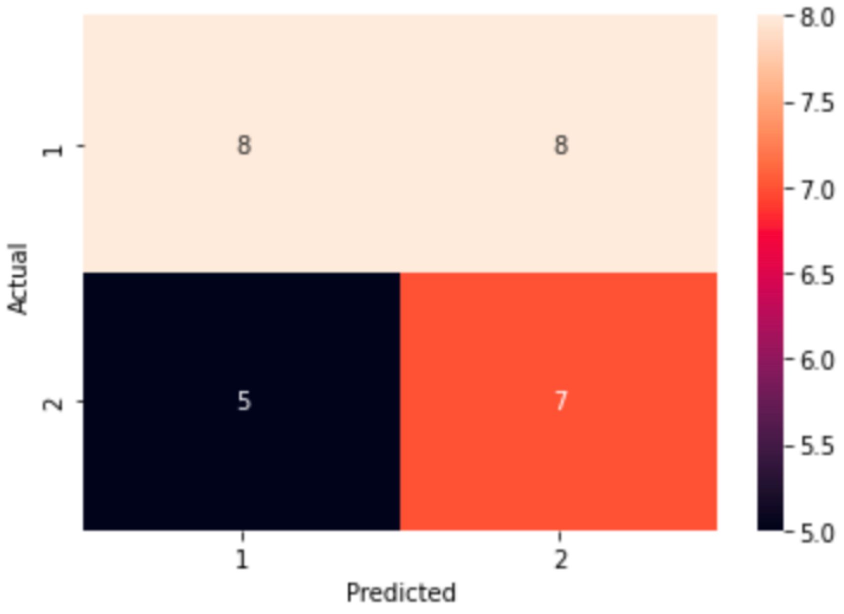
Logistic Regression is a supervised learning algorithm that works on a binary variable to predict the probability of either 0 or 1. The algorithm is based on the famous S-shaped curve Logistic function that models population growth and can take any real number and map it within the 0 and 1 range. It is a simple yet effective classification algorithm that has been used quite extensively in the credit scoring, medical, and gaming industries.

Logistic Regression is also popular in BCI applications as they mostly deal with binary classification problems. The algorithm is also used for [5] feature selection for EEG-based Motor Imagery Classification. The feature selection performed using Logistic Regression yielded higher accuracy of 95.21% for BCI competition III-Dataset and 94.83% for BCI competition IV-Dataset 2a. These are very high results and make Logistic Regression a promising classifier for BCI applications.

For our project, we used Logistic Regression with five-fold cross-validation. There are five different solvers to choose from and we used the default lbfgs which is ideal for multiclass problems. GridSearch was also used with Logistic Regression to drive the best parameters for better accuracy. The following table shows the accuracies we achieved first with cross-validation followed by tuned parameters and finally on unseen test data.

Cross Validation Accuracy (Training data)	Cross Validation Accuracy (Training data with tuned parameters)	Accuracy on Unseen Data
50%	50%	60%

Table 3.3 Logistic Regression cross-validation accuracy



Confusion Matrix – Logistic Regression

3.4.4 KNN

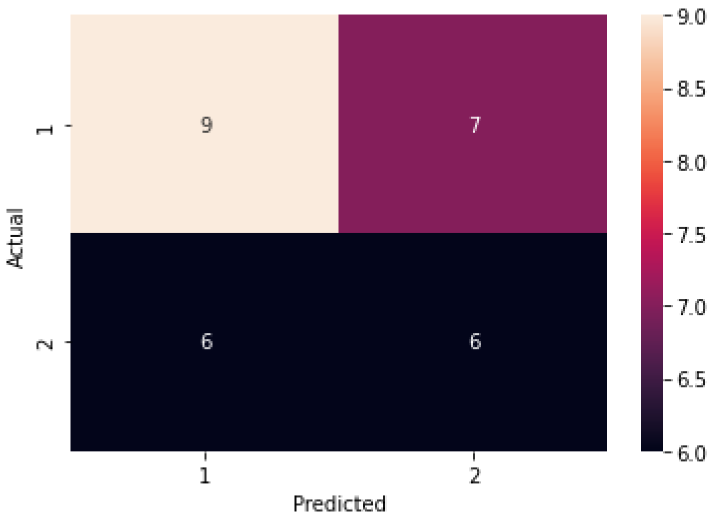
KNN can be used both in classification as well as regression problems. KNN works by assuming that the k number of nearest neighbors belong to the same class therefore when a new data point is predicted, the nearest neighbor's class is assigned to it. It's a simple algorithm that doesn't need to make any assumptions or significant tuning of parameters. It works well on small datasets and is not known for good performance on larger datasets.
For our project we have used KNN with 2-fold cross validation. For our baseline model, we constructed a pipeline with standard scalar and KNN classifier and computed cross-validation scores by training our model on different training sets and testing on validation sets.
To make further improvements to our baseline model, we performed GridSearchCV along with cross-validation to find the optimal parameters. The parameter Grid used is as follows:
leaf_size=list(range(1,30))
n_neighbors=list(range(1,50))
P=[1,2]
Weights=['uniform' , 'distance']
After obtaining the optimal parameters we again computed the cross-validation accuracy score but this time with the best parameters.
Finally, we predict the values for the hidden data set using our model trained on best parameters.
The accuracy scores obtained by performing cross-validation, cross-validation with

tuned parameters, and predicting values for unseen test data are as follows:

Cross Validation Accuracy (Training data)	Cross Validation Accuracy (Training data with tuned parameters)	Accuracy on Unseen Data
38 %	52%	36%

Table 3.4 KNN Accuracy Table

As seen in table 3.4, our model's accuracy is increased by 14 percent when it is trained using tuned parameters. Accuracy drops on the test set therefore our model is overfitting.



Confusion Matrix – KNN

3.4.5 XGBoost

XGBoost is a more advanced and efficient version of gradient boosting. Due to parallel computations, it has a fast-learning ability providing both linear and tree learning models [22]. XGBoost is not used significantly in EEG related works but we are going to explore it in our work.
For our project we have used XGBoost with 5-fold cross validation. For our baseline model, we constructed a pipeline with standard scalar and XGBoost classifier and computed cross-validation scores by training our model on different training sets and testing on validation sets.
To make further improvements to our baseline model, we performed GridSearchCV along with cross-validation to find the optimal parameters. The parameter Grid used is as follows:

- learning_rate=[0.01, 0.1, 1.0]
- max_depth=[3,12,25]
- max_features=["log2","sqrt"]

- n_estimators=[1,25,100]

After obtaining the optimal parameters we again computed the cross-validation accuracy score but this time with the best parameters.

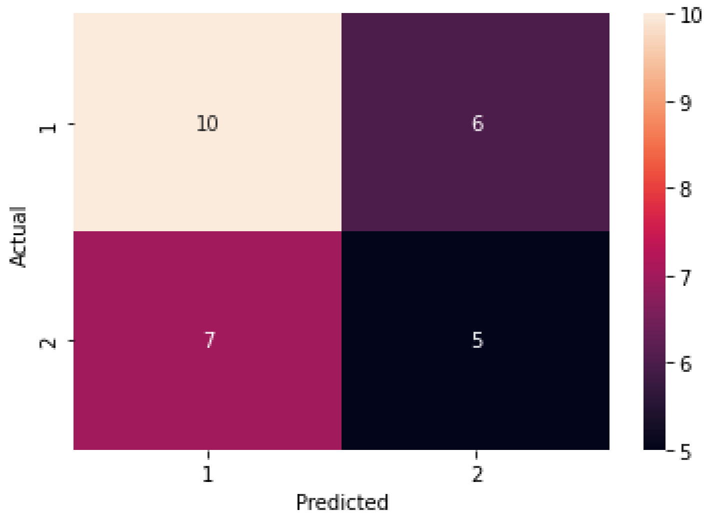
Finally, we predict the values for the hidden data set using our model trained on best parameters.

The accuracy scores obtained by performing cross-validation, cross-validation with tuned parameters, and predicting values for unseen test data are as follows:

Cross Validation Accuracy (Training data)	Cross Validation Accuracy (Training data with tuned parameters)	Accuracy on Unseen Data
58%	60%	71%

Table 3.5 XGBoost Accuracy Table

Table 3.5 shows that our model performed well on training data but it performs exceptionally well on the test data.



Confusion Matrix – XGBoost

3.4.6 Adaboost:

For classification problems adaptive booting is a useful ensemble technique. The weight-assigning method of adaboost after each iteration is the strongest feature of adaboost which makes this model unique from others. There are basically three main steps in adaboosting algorithm:

- Sampling
- Training
- Combination

In **Sampling**, different samples are selected based on replacement. From the start, all the selected samples have equal weights. After each iteration weights are updated and at the end of the sampling step normalization is performed to normalize the updated weights.

In **Training** step now using these selected samples week learners and classifiers are used and according to training error this algorithm includes or exclude the learners.

In **the Combination** or test phase all the classifiers are combined to predict the value of the new sample and give us the final classification model.

So, this algorithm starts from the base estimator and calculates the weights and weights of misclassified samples are then adjusted correctly which makes the adaboost algorithm different from others. This is a supervised machine learning algorithm.

There are the following parameters of adaboost which are shown below:

- base_estimator
- n_estimators
- learning_rate
- algorithm
- random_state

The base_estimator is from where boosted ensemble grows. If the base estimator is none then it initializes with decision tree classifier with maximum depth of 1.

The n_estimators is the number where the boosting is terminated after implementing maximum estimators. By default, its value is 50.

The learning_rate is the weight applied after every iteration to any classifier. The contributions of classifiers are increased by increasing learning rate, but we have been a trade-off between n_estimators and learning_rate.

By default, SAMME.R algorithm is used which is real boosting. And this achieves high results in fewer iterations and converges faster than SAMME.

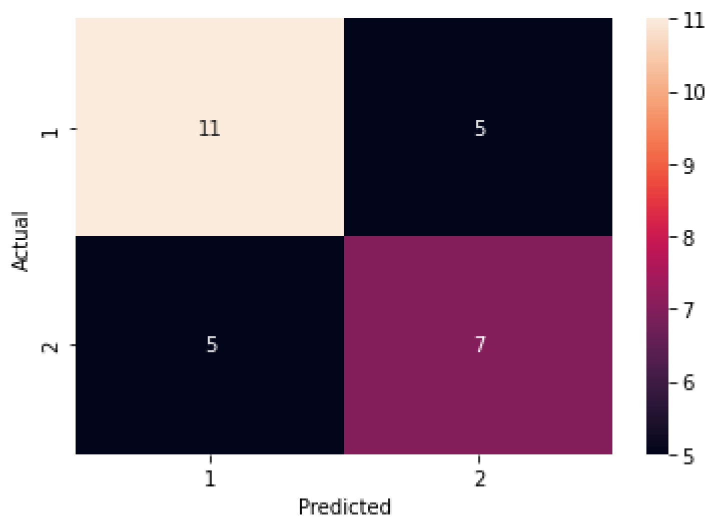
The random_state is by default none and at each iteration this control randomness.

So, adaboost is widely used in BCI applications. For our motor imaginary left and right-hand movement classification we also use the adaboost algorithm. BCI is a brain computer interface from which we can interact with the outer environment and in our system from BCI data we control the movement of left and right hands. So, it is simply a classification problem, and we also use the adaboost algorithm as well to check whether this machine learning model is well or not for our classification problem. This model performs well when we extract the features of 'app_entropy' and 'time_corr' which is time correlation from the BCI dataset.

The results for this model on training and testing is given below:

Cross Validation Accuracy (Training data)	Cross Validation Accuracy (Training data with tuned parameters)	Accuracy on Unseen Data
60.8%	60.8%	67.9%

Table 3.6 AdaBoost cross-validation accuracy



Confusion Matrix –AdaBoost

3.4.7 Decision Tree:

Decision tree is a non-parametric supervised learning algorithm which is used for regression and classification. Decision tree simply defines learning decision rules from the extracted features and predicts the target class.
Decision tree used the rule-based approach, but different classification algorithms use probability approaches like Naive Bayes. The idea behind Decision Trees is to build yes/no questions using dataset attributes and then split the dataset until all data points belonging to each class are isolated.
You're putting the data into a tree structure with this method. You're adding a node to the tree every time you ask a question. The root node is the very first node. The outcome of asking a query divides the dataset into new nodes based on the value of a feature. The last nodes formed if you decide to halt the process after a split are known as leaf nodes.
By learning basic decision rules inferred from past data, the purpose of employing a Decision Tree is to develop a training model that can be used to predict the class or value of the target variable (training data). We start at the root of the tree when using Decision Trees to forecast a class label for a record. The values of the root attribute and the record's attribute are compared. We follow the branch that corresponds to that value and go to the next node based on the comparison.
Based on target data there are two types of decision trees which are defined as:

- Categorical variable decision tree
- Continuous variable decision tree

The decision to make strategic splits has a significant impact on a tree's accuracy. The decision criteria for classification and regression trees are different. To decide whether to break a node into two or more sub-nodes, decision trees employ a variety of

techniques. The homogeneity of the generated sub-nodes improves with the generation of sub-nodes. To put it another way, the purity of the node improves as the target variable grows. The decision tree divides the nodes into sub-nodes based on all available factors, then chooses the split that produces the most homogenous sub-nodes.
The following are some of the benefits of decision trees:

- It is easy to comprehend and interpret. It is possible to visualize trees.
- The cost of utilizing the tree (that is, predicting data) is proportional to the amount of data points needed to train it.
- Capable of dealing with difficulties with several outputs.
- The model is based on a white box. If a circumstance can be observed in a model, Boolean logic can simply describe the situation. In contrast, the findings of a black box model (such as an artificial neural network) may be more difficult to decipher.
- Statistical tests can be used to validate a model. As a result, the model's dependability may be accounted for.

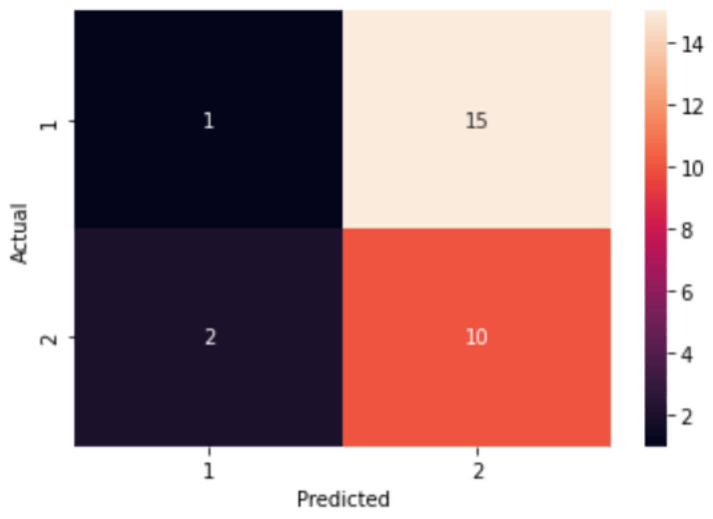
There are also some drawbacks of the decision tree as well which are listed below:

- Some topics, such as XOR, parity, and multiplexer difficulties, are difficult to grasp because decision trees do not clearly describe them.
- Because slight changes in the data might result in an entirely different tree being constructed, decision trees can be unstable. The use of decision trees within an ensemble helps to solve this difficulty.
- If some classes dominate, decision tree learners build biased trees. It is consequently advised that the dataset be balanced before using the decision tree to suit it.

We use the decision tree for our classification problem as well. For our classifier we use the gini criterion, random state of 100, maximum depth of 3 and minimum sample leaf of 5. On BCI dataset in our imaginary system for classification this perform well on selected features of ‘app_entropy’, ‘quantile’, ‘time_corr’ and ‘mean’.
Because we test our data on different machine learning classifiers to check which classifiers perform well on that dataset. Because in machine learning classification if on some datasets some machine learning classifiers perform better then maybe on other datasets some others perform better. So, in our case the results for training and testing data after the train test split are shown below:

Cross Validation Accuracy (Training data)	Cross Validation Accuracy (Training data with tuned parameters)	Accuracy on Unseen Data
49.1%	50.9%	39.3%

Table 3.7 Decision Tree cross-validation accuracy



Confusion Matrix – Decision Tree

3.4.8 ANN or MLP:

ANN simply defined as artificial neural network and MLP means multi-layer perceptron. The main difference between machine learning and deep learning is that for machine learning models we must do the feature extraction and these features are given to machine learning models but in deep learning this task of feature engineering is done by deep learning models.
At each layer of an Artificial Neural Network, or ANN, there are many perceptrons/neurons. Because inputs are exclusively processed in the forward direction, an ANN is also known as a Feed-Forward Neural Network.
There are different types of layers in deep learning models which are given below:

- Input layer
- Hidden layers
- Output layers

ANN can be used to solve problems related to image data, tabular data and text data. Any nonlinear function may be learned by an Artificial Neural Network. As a result, these networks are commonly referred to as Universal Function Approximators. ANNs can learn weights that map any input to the desired output. The activation function is one of the key reasons for universal approximation. The network's nonlinear features are introduced using activation functions. This enables the network to learn any complicated input-output relationship.
Sequential information in the input data is not captured by ANN, which is essential when dealing with sequence data.

Deep neural networks are easily implemented using keras and tenser flow. For our BCI problem and for the left- and right-hand classification we also implemented the artificial

neural network which is a sequential model.
We add the following layers in the deep neural network:

- Input layer
- Hidden layer with relu as activation function
- Batch Normalization Layer
- LeakyRelu layer
- Hidden layer with 5 units
- Batch Normalization Layer
- LeakyRelu layer
- Dropout Layer
- Output layer with one neuron

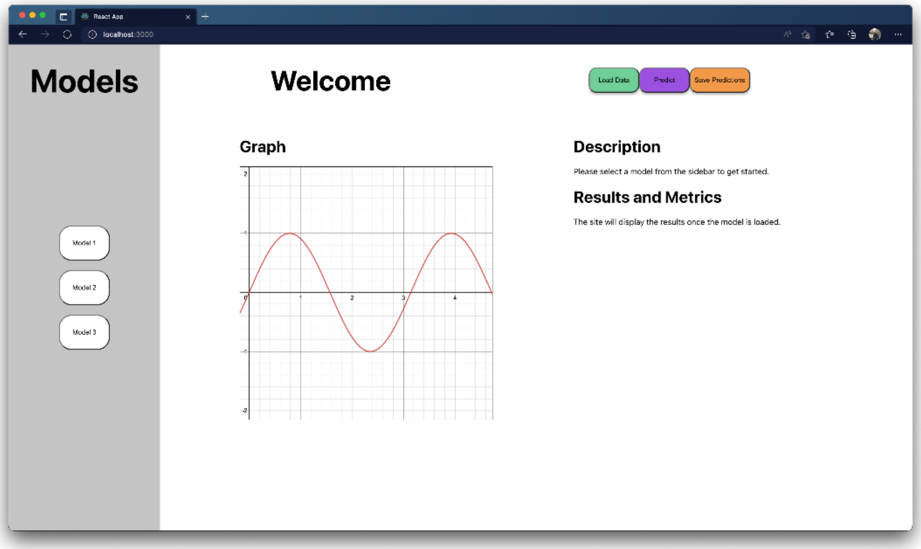
In our case this structure gives us maximum accuracy of 48.2 percent on training data. To tune the artificial neural networks is a difficult job but for our dataset which we achieve is maximum 48 to 49 percent. But in future if someone works properly in it and spends more time on that above 75 percent accuracy can be achieved. According to my knowledge because the samples in the dataset are very low that's why we could not professionally trained the models but in future if in dataset there are more entries or there are more samples in the dataset then may be higher accuracy can be achieved.

3.4.9 CNN:

Convolutional Neural Network is a type of Deep Learning Model. They are widely used for EEG classification purposes due to its ability of extracting features without having to apply any external Feature Extractor. This is done by the 'Max Pooling' layer in the network.
A CNN takes the input signal, and it multiplies the data with the 'Kernel' matrix present in the Convolutional Layer. This is followed by the 'Max Pooling' Layer and 'Flattening' Layer. These are followed by a series of 'Dense' layers which performs the task of Learning. For our implementation, we have applied a 'softmax' activation function as this allows our model to perform the task of classification.
We are getting an accuracy of 100% which implies that our model has overfit the data. This could be attributed to the fact that we only have 140 trials of data and our model is learning the data too well.

3.5 GUI and Backend

The GUI was implemented using the React framework. This is a component-based framework, which means that the view of the web page is composed by smaller reusable components. The two main components in the page are the 'Sidebar' component and the 'ModelView' component. From there the rest of smaller components fall into place.



Description

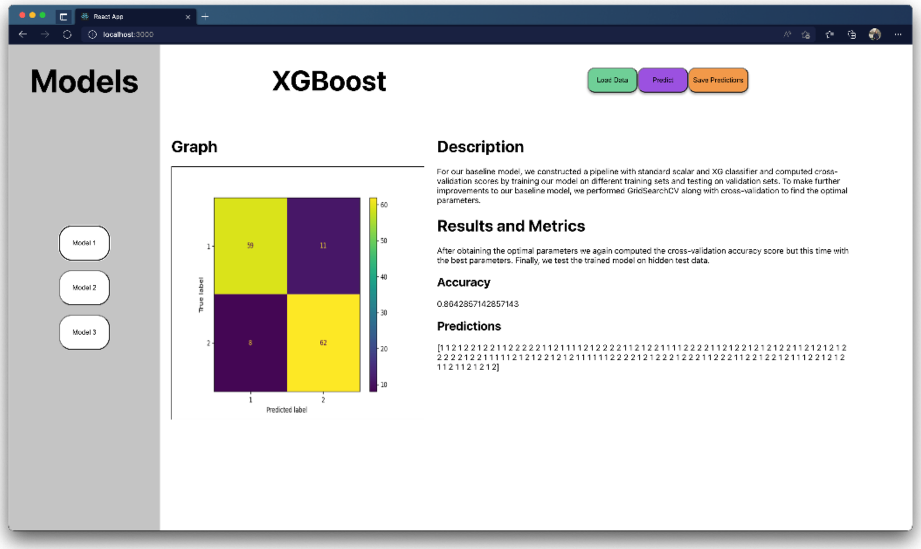
Please select a model from the sidebar to get started.

Results and Metrics

The site will display the results once the model is loaded.

Image 3.14: GUI Welcome Page

Once the user selects a model from the sidebar, the GUI sends a request to the backend and displays information of the selected model, as well as a confusion matrix of the results.



Description

For our baseline model, we constructed a pipeline with standard scalar and XG classifier and computed cross-validation scores by training our model on different training sets and testing on validation sets. To make further improvements to our baseline model, we performed GridSearchCV along with cross-validation to find the optimal parameters.

Results and Metrics

After obtaining the optimal parameters we again computed the cross-validation accuracy score but this time with the best parameters. Finally, we test the trained model on hidden test data.

Accuracy

0.8642857142857143

Predictions

```
[1 1 2 1 2 2 1 2 2 1 1 2 2 2 2 2 1 1 2 1 1 1 1 2 1 2 2 2 2 1 1 2 1 2 2 1 1 1 1 2 2 2 2 1 1 2 1 2 2 1 2 1 2 1 2 2 1 1 2 1 2 1 2 1 2
2 2 2 2 1 2 2 1 1 1 1 1 2 1 2 1 2 2 1 2 1 2 1 1 1 1 1 2 2 2 2 1 2 1 2 2 2 1 2 2 2 1 1 2 2 2 1 1 2 2 1 2 2 1 2 1 1 1 2 2 1 2 1 2
1 1 2 1 1 2 1 2 1 2]
```

Image 3.15: GUI Model View

The backend is mainly on the ‘app.js’ file in the backend folder. From here the backend access the desired model, loads it, sends the data and returns the predictions.

4. Experimental Results

4.1 Combination of MNE Feature

We applied the feature extraction and looping all possible combinations to the classifier mentioned in 3.4. The best two combinations for logistic regression are standard deviation, approximate entropy. For random forest and gradient boosting have the same best feature, which is quantile and approximate entropy. The score for each model is present in the table below.

Classifier	Best features
1. Random Forest	Approximate entropy, Quantile, Mean, Correlation Coefficients (Time Domain)
2. Logistic Regression	Approximate entropy and Standard deviation
3. Support Vector Machine	Approximate entropy, Correlation Coefficients (Time Domain)
4. KNN	
5. XGBoost	
6. Decision Tree	
7. AdaBoost	

Table 4.1 Best MNE features for classifiers

4.2 Machine Learning Scores

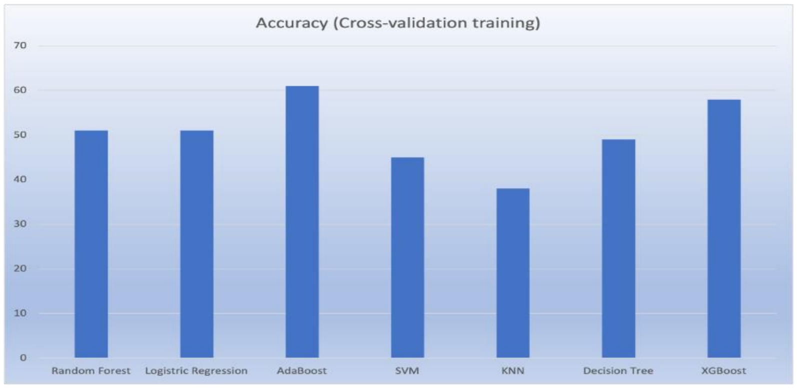


Image 4.1 Cross-validation accuracy on training data

AdaBoost achieved the highest accuracy of cross validation on training data followed by XGBoost and Random Forest and Logistic Regression.

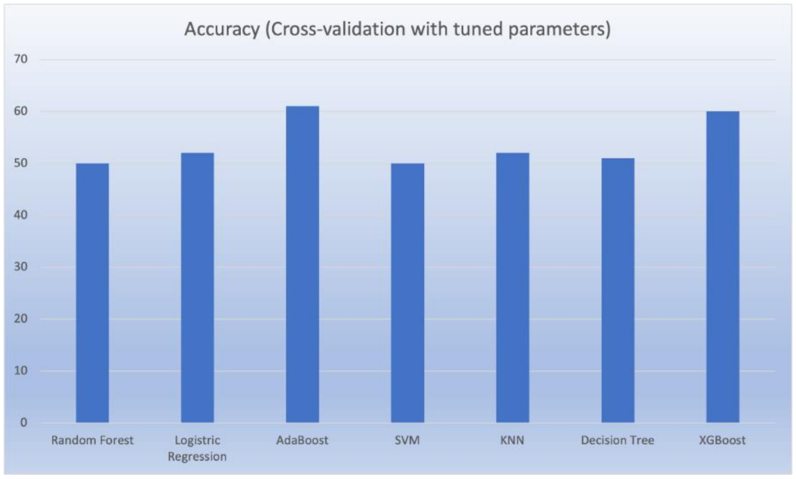


Image 4.2 Cross-validation accuracy with tuned parameters

After tuning the parameters using GridSearch, the accuracy of almost all the classifiers improved except for the XGBoost which stayed at 61%.

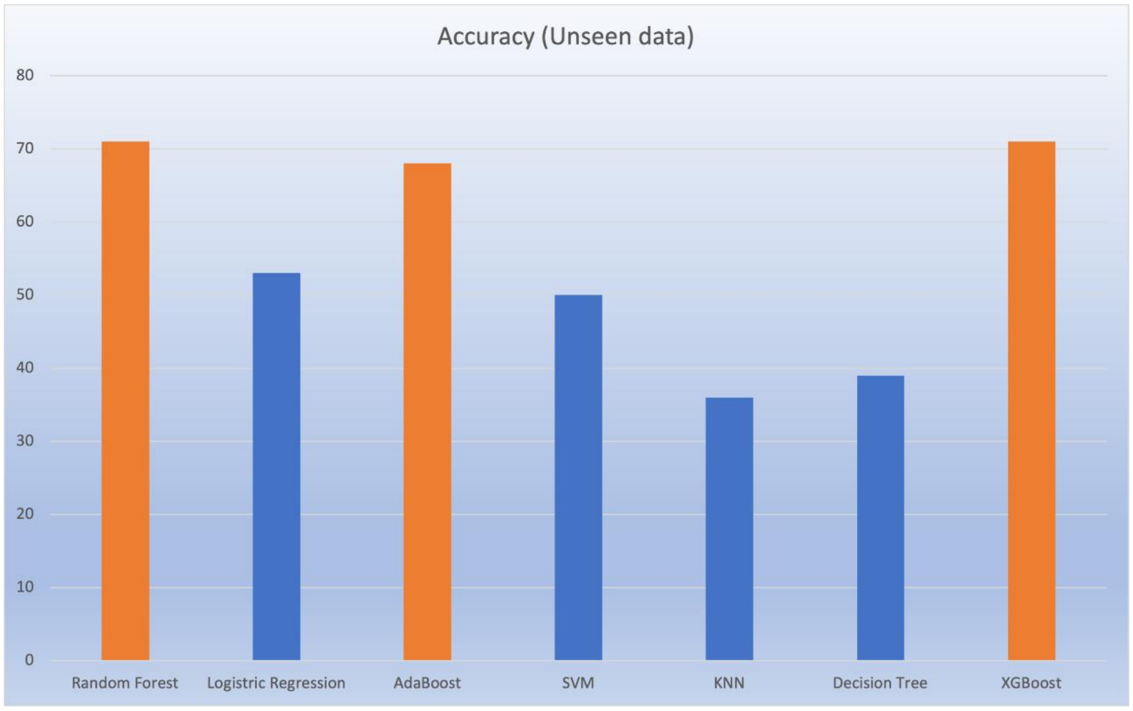


Image 4.3 Accuracy on unseen data

Classifiers with the highest accuracy on unseen data were Random Forest, XGBoost, and AdaBoost. We incorporated these classifiers into our GUI.

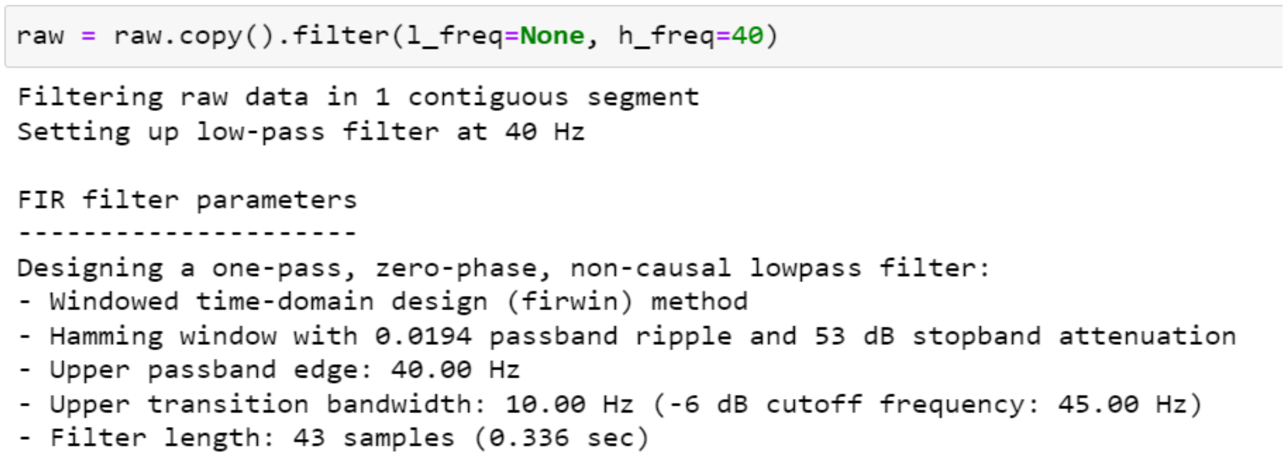
5. Testing

5.1 Testing methods

Testing was based on the following techniques:

5.1.1 Unit Testing

- Each modules were tested independently to ensure that they were functional and as per requirements. This was mainly done using White Box Testing.
- For the Data-Preprocessing module, checks were made to ensure Signal Filtering, Feature Selection and Feature-Extraction were performed correctly using MNE-Python [6].

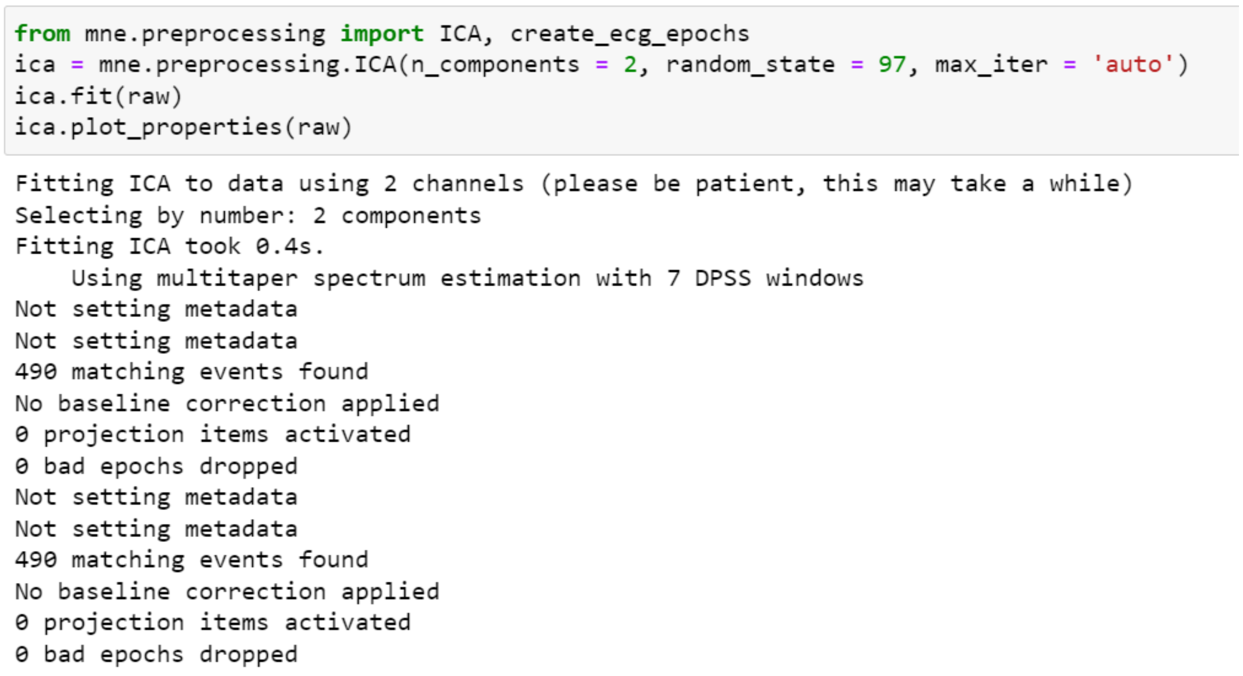


```
raw = raw.copy().filter(l_freq=None, h_freq=40)
```

```
Filtering raw data in 1 contiguous segment
Setting up low-pass filter at 40 Hz

FIR filter parameters
---------------------
Designing a one-pass, zero-phase, non-causal lowpass filter:
- Windowed time-domain design (firwin) method
- Hamming window with 0.0194 passband ripple and 53 dB stopband attenuation
- Upper passband edge: 40.00 Hz
- Upper transition bandwidth: 10.00 Hz (-6 dB cutoff frequency: 45.00 Hz)
- Filter length: 43 samples (0.336 sec)
```

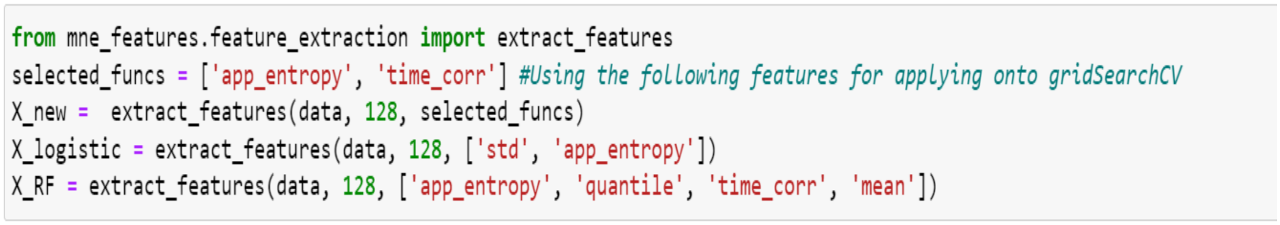
Image 5.1: Unit Testing – Filtering Signal



```
from mne.preprocessing import ICA, create_ecg_epochs
ica = mne.preprocessing.ICA(n_components = 2, random_state = 97, max_iter = 'auto')
ica.fit(raw)
ica.plot_properties(raw)
```

```
Fitting ICA to data using 2 channels (please be patient, this may take a while)
Selecting by number: 2 components
Fitting ICA took 0.4s.
    Using multitaper spectrum estimation with 7 DPSS windows
Not setting metadata
Not setting metadata
490 matching events found
No baseline correction applied
0 projection items activated
0 bad epochs dropped
Not setting metadata
Not setting metadata
490 matching events found
No baseline correction applied
0 projection items activated
0 bad epochs dropped
```

Image 5.2: Unit Testing – Feature Selection



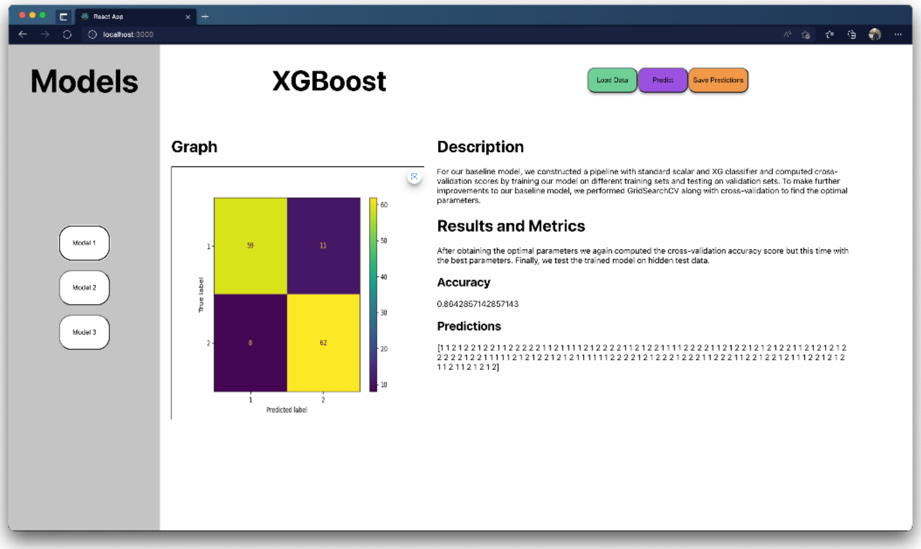
```
from mne_features.feature_extraction import extract_features
selected_funcs = ['app_entropy', 'time_corr'] #Using the following features for applying onto gridSearchCV
X_new =  extract_features(data, 128, selected_funcs)
X_logistic = extract_features(data, 128, ['std', 'app_entropy'])
X_RF = extract_features(data, 128, ['app_entropy', 'quantile', 'time_corr', 'mean'])
```

Image 5.3: Unit Testing – Feature Extraction

- For the Machine Learning module, it was ensured that we tried implementing more than 5 Traditional Machine Learning Models and 2 Deep Learning Models. It was ensured that each models were implemented accurately and we get accuracy scores for each. It was also made certain that we had atleast 3 models giving an accuracy score of above 65% for unseen test data. Metrices used for evaluating were accuracy score and confusion matrix.
- For the GUI module, tests were performed to ensure it was functional.

5.1.2 Integration Testing

The 3 different modules were integrated, and System Integration tests were performed to ensure the results of all models were displayed on the GUI.



Description

For our baseline model, we constructed a pipeline with standard scalar and XG classifier and computed cross-validation scores by training our model on different training sets and testing on validation sets. To make further improvements to our baseline model, we performed GridSearchCV along with cross-validation to find the optimal parameters.

Results and Metrics

After obtaining the optimal parameters we again computed the cross-validation accuracy score but this time with the best parameters. Finally, we test the trained model on hidden test data.

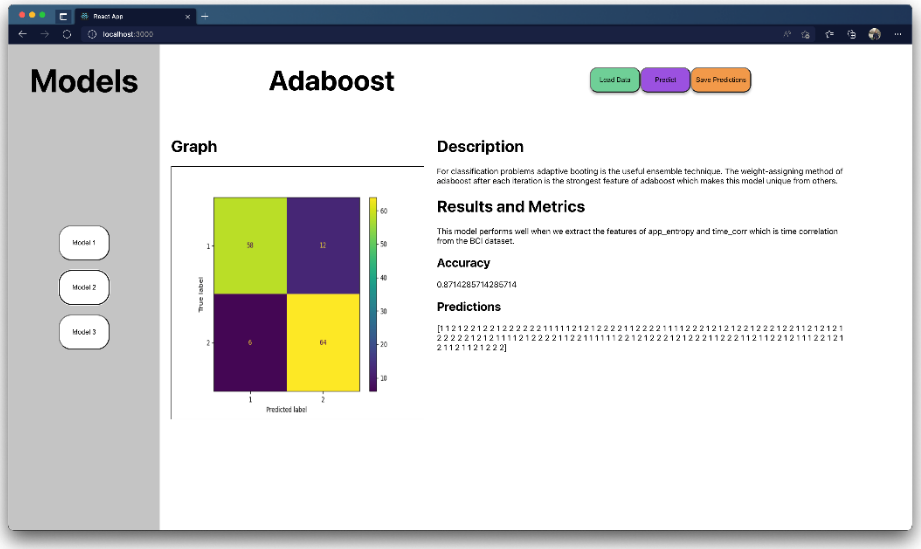
Accuracy

0.8642857142857143

Predictions

```
[1 1 2 1 2 2 1 2 2 1 1 2 2 2 2 2 1 1 2 1 1 1 1 2 1 2 2 2 2 1 1 2 1 2 2 1 1 1 1 2 2 2 2 1 1 2 1 2 2 1 2 1 2 1 2 2 1 1 2 1 2 1 2 1 2
2 2 2 2 1 2 2 2 1 1 1 1 1 2 1 2 1 2 2 1 2 1 2 1 2 1 1 1 1 1 1 2 2 2 2 1 2 1 2 2 2 1 2 2 2 1 1 2 2 2 1 1 2 2 1 2 2 1 2 1 1 1 2 2 1 2 1 2 1 2
1 1 2 1 1 2 1 2 1 2]
```

Image 5.4: Testing Model 1



Description

For classification problems adaptive booting is the useful ensemble technique. The weight-assigning method of adaboost after each iteration is the strongest feature of adaboost which makes this model unique from others.

Results and Metrics

This model performs well when we extract the features of app_entropy and time_corr which is time correlation from the BCI dataset.

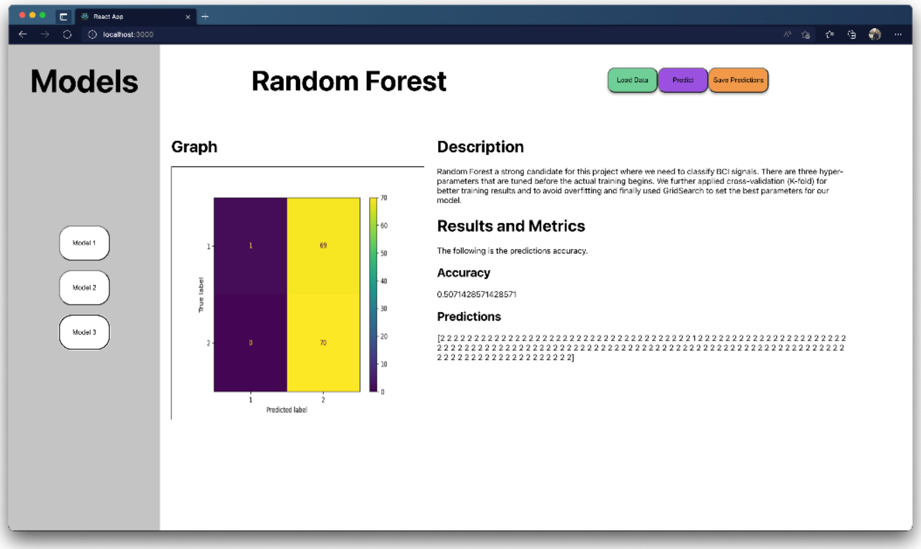
Accuracy

0.8714285714285714

Predictions

```
[1 1 2 1 2 2 1 2 2 1 2 2 2 2 2 2 1 1 1 1 1 2 1 2 1 2 2 2 2 1 1 2 2 2 2 1 1 1 1 2 2 2 1 2 1 2 1 2 2 1 2 2 2 1 2 2 1 1 2 1 2 1 2 1
2 2 2 2 2 1 2 1 2 1 1 1 1 2 1 2 2 2 2 1 1 2 2 1 1 1 1 1 2 2 1 2 1 2 2 2 1 2 1 2 2 2 1 1 2 2 2 1 1 2 1 1 2 2 1 2 1 1 1 2 2 1 2 1
2 1 1 2 1 1 2 1 2 2 2]
```

Image 5.5: Testing Model 2



Random Forest

Graph

Description

Random Forest a strong candidate for this project where we need to classify BCI signals. There are three hyper-parameters that are tuned before the actual training begins. We further applied cross-validation (K-fold) for better training results and to avoid overfitting and finally used GridSearch to set the best parameters for our model.

Results and Metrics

The following is the predictions accuracy.

Accuracy

0.5071428571428571

Predictions

```
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

Image 5.6: Testing Model 3

5.1.3 User-Acceptance Testing

Black Box testing was performed to ensure user was able to load different models and the unseen test accuracy along with the predictions were displayed to the user. Use case diagram for testing from User's point of view is shown in Image 5.7

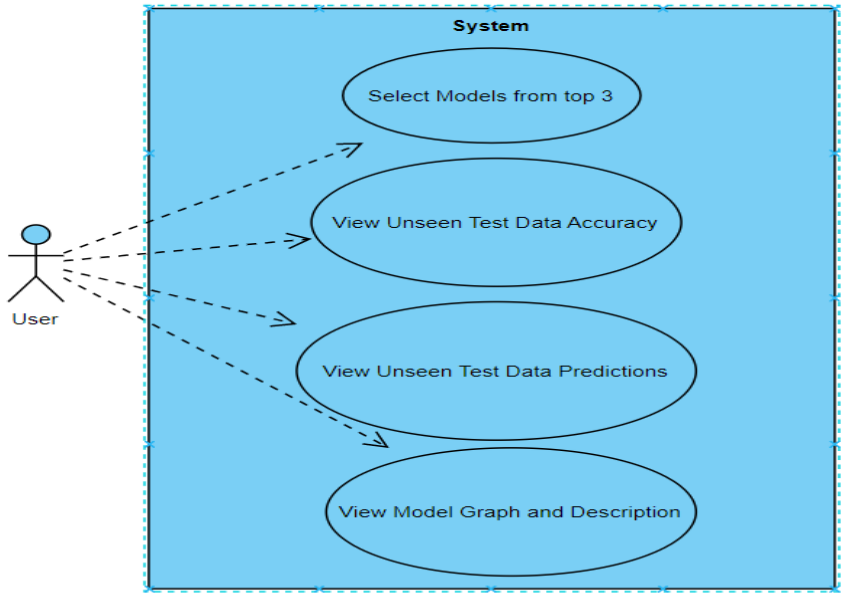

System
Select Models from top 3
View Unseen Test Data Accuracy
View Unseen Test Data Predictions
View Model Graph and Description
User

Image 5.7: Black Box Testing – Use Cases

6. Conclusion

This project was developed based on the BCI Competition II (2003) Dataset 3 [5]. Techniques of Signal Filtering, Feature Selection and Feature Extraction were implemented to process the signal data. However, our models were only able to achieve an unseen validation accuracy of 71%. This could be attributed to the fact that we only had 140 trials of data for training our models. Moreover, the data was split into training and testing set resulting in only 112 trials for training our models. This accuracy could have been improved if the number of trials were increased.

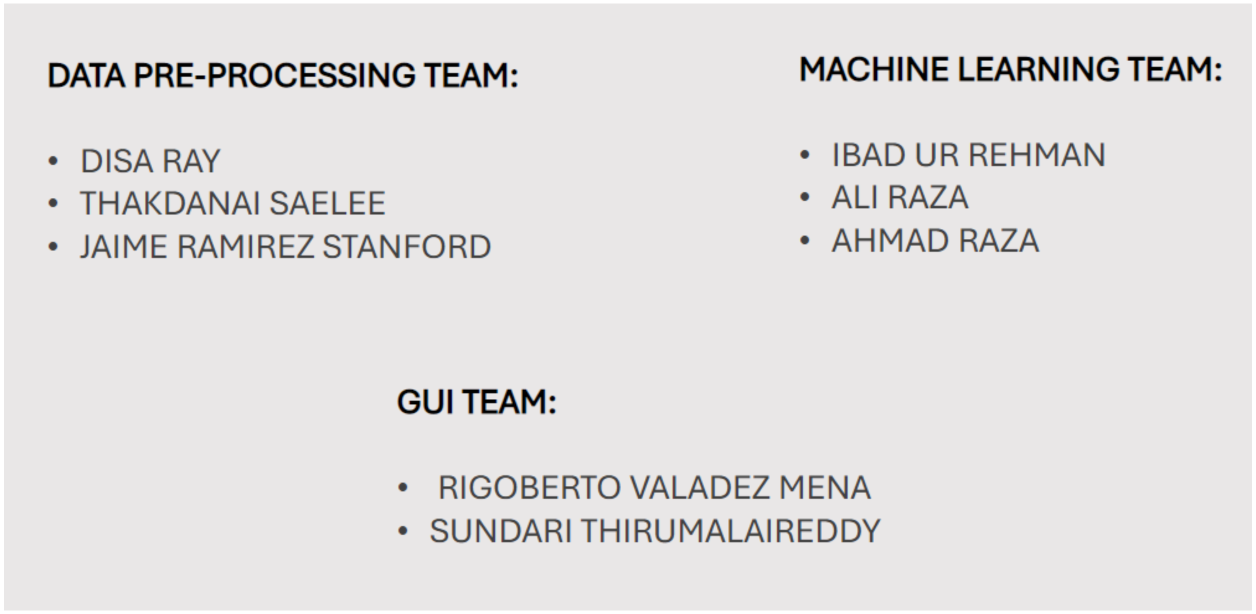
Moreover, few features in our GUI could not be implemented due to unsupported browser issues. Thus, for future these features could be implemented to ensure a fully working system which allows user to classify left- and right-hand imagery movements.

Furthermore, different other techniques related to Filtering, Feature Selection and Feature Extraction could be used to improve accuracy scores of models.

7. Project Management

The project was entirely prepared following agile methodology. We used GitLab to collaborate code and Jira to work on our Project Backlog.

7.1 Team Structure



DATA PRE-PROCESSING TEAM:

- DISA RAY
- THAKDANAI SAELEE
- JAIME RAMIREZ STANFORD

MACHINE LEARNING TEAM:

- IBAD UR REHMAN
- ALI RAZA
- AHMAD RAZA

GUI TEAM:

- RIGOBERTO VALADEZ MENA
- SUNDARI THIRUMALAIREDDY

Image 7.1: Team Division Structure

7.2 Work Breakdown Structure

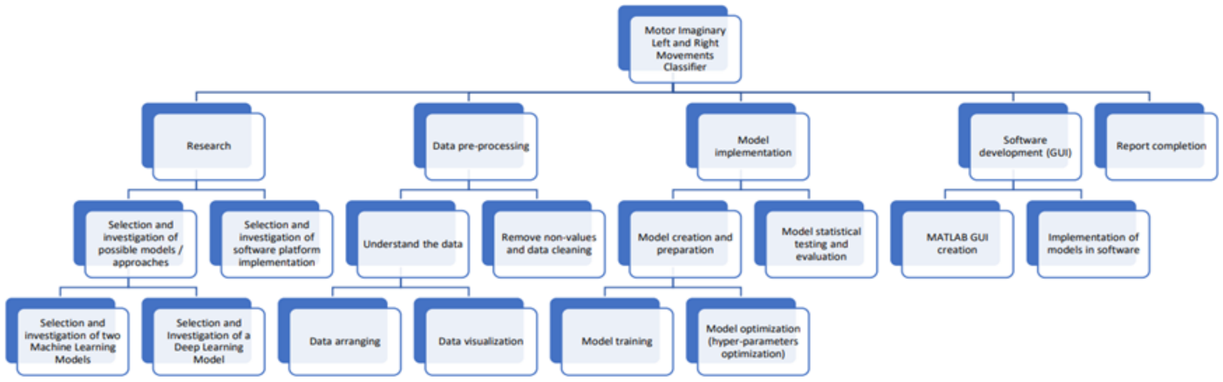
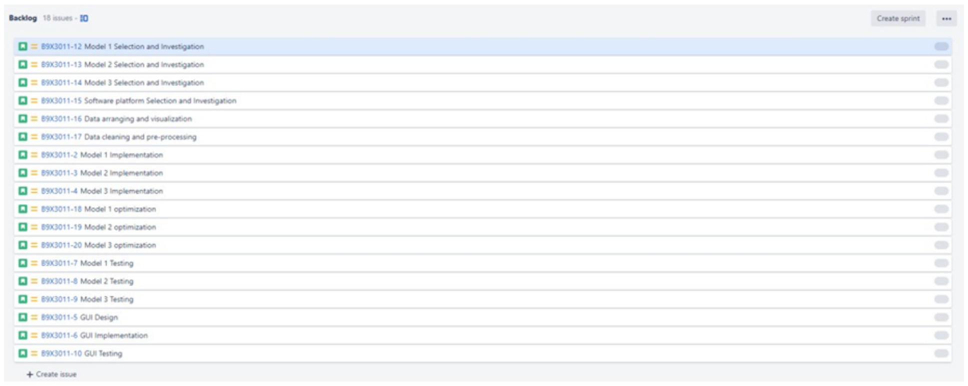

Motor Imaginary Left and Right Movements Classifier
Research
Data pre-processing
Model implementation
Software development (GUI)
Report completion
Selection and investigation of possible models / approaches
Selection and investigation of software platform implementation
Understand the data
Remove non-values and data cleaning
Model creation and preparation
Model statistical testing and evaluation
MATLAB GUI creation
Implementation of models in software
Selection and investigation of two Machine Learning Models
Selection and Investigation of a Deep Learning Model
Data arranging
Data visualization
Model training
Model optimization (hyper-parameters optimization)

Image 7.2 Work Breakdown Structure [26]

7.3 Jira Backlog



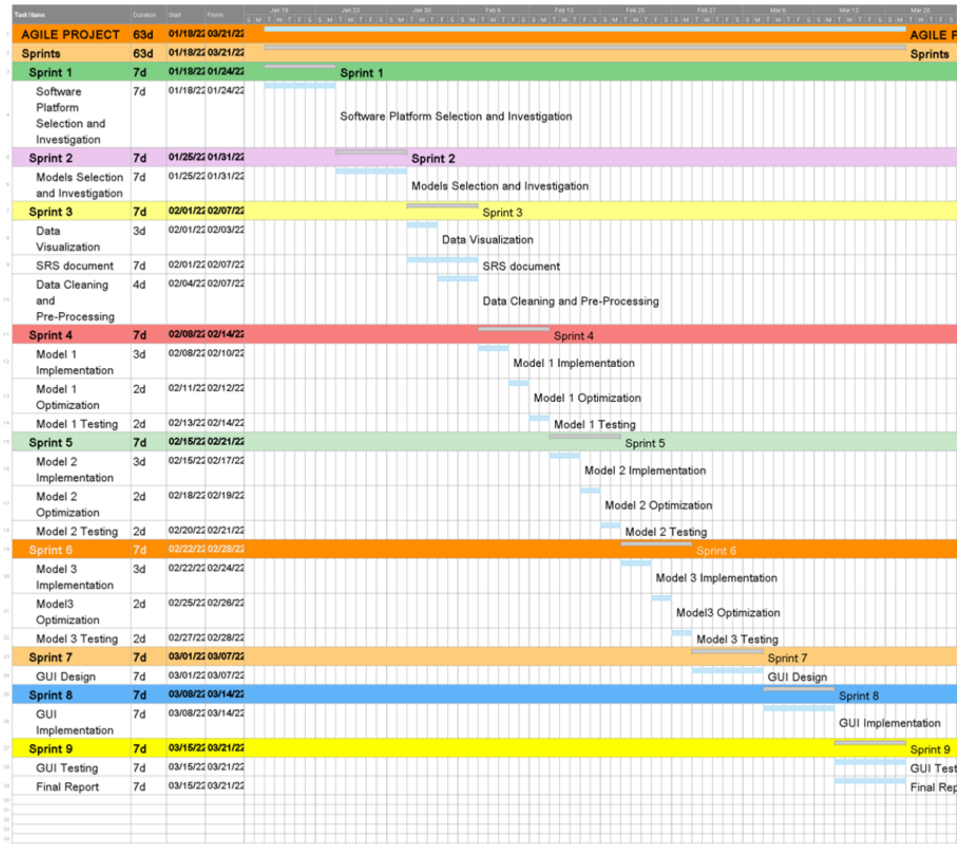
Backlog 18 issues - 0

B9X3011-12 Model 1 Selection and Investigation
B9X3011-13 Model 2 Selection and Investigation
B9X3011-14 Model 3 Selection and Investigation
B9X3011-15 Software platform Selection and Investigation
B9X3011-16 Data arranging and visualization
B9X3011-17 Data cleaning and pre-processing
B9X3011-2 Model 1 Implementation
B9X3011-3 Model 2 Implementation
B9X3011-4 Model 3 Implementation
B9X3011-18 Model 1 optimization
B9X3011-19 Model 2 optimization
B9X3011-20 Model 3 optimization
B9X3011-7 Model 1 Testing
B9X3011-8 Model 2 Testing
B9X3011-9 Model 3 Testing
B9X3011-5 GUI Design
B9X3011-6 GUI Implementation
B9X3011-10 GUI Testing

+ Create issue

Image 7.3: Jira Backlog [26]

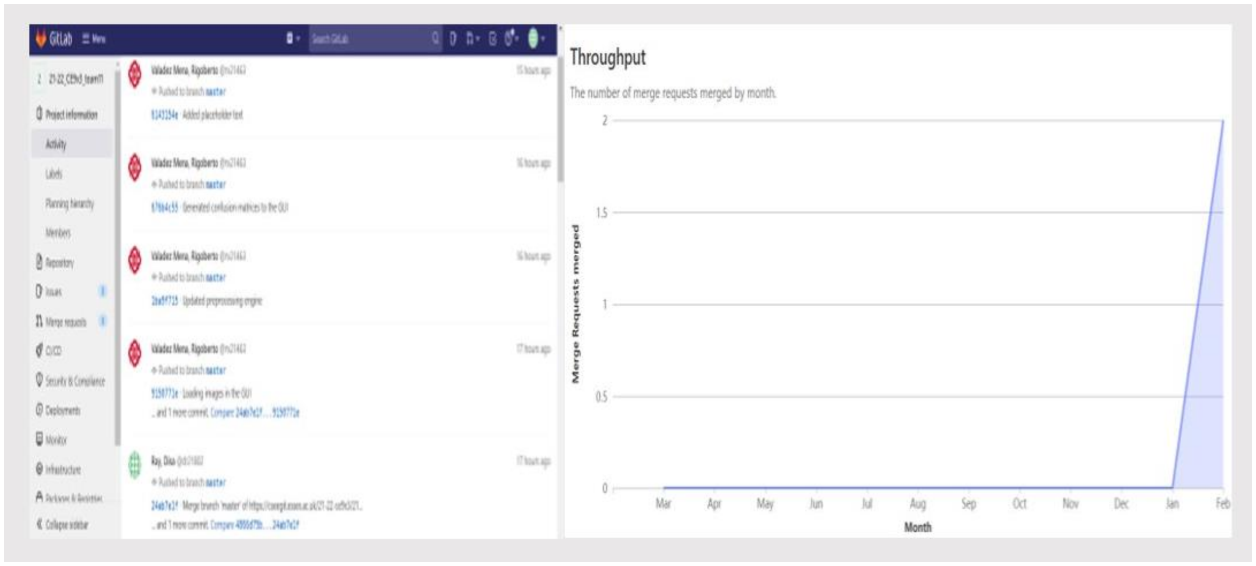
7.4 Gantt chart with milestones for activity network



Task Name	Duration	Start	Finish
AGILE PROJECT	**63d**	**01/18/22**	**03/21/22**
Sprints	**63d**	**01/18/22**	**03/21/22**
Sprint 1	**7d**	**01/18/22**	**01/24/22**
Software Platform Selection and Investigation	7d	01/18/22	01/24/22
Sprint 2	**7d**	**01/25/22**	**01/31/22**
Models Selection and Investigation	7d	01/25/22	01/31/22
Sprint 3	**7d**	**02/01/22**	**02/07/22**
Data Visualization	3d	02/01/22	02/03/22
SRS document	7d	02/01/22	02/07/22
Data Cleaning and Pre-Processing	4d	02/04/22	02/07/22
Sprint 4	**7d**	**02/08/22**	**02/14/22**
Model 1 Implementation	3d	02/08/22	02/10/22
Model 1 Optimization	2d	02/11/22	02/12/22
Model 1 Testing	2d	02/13/22	02/14/22
Sprint 5	**7d**	**02/15/22**	**02/21/22**
Model 2 Implementation	3d	02/15/22	02/17/22
Model 2 Optimization	2d	02/18/22	02/19/22
Model 2 Testing	2d	02/20/22	02/21/22
Sprint 6	**7d**	**02/22/22**	**02/28/22**
Model 3 Implementation	3d	02/22/22	02/24/22
Model3 Optimization	2d	02/25/22	02/26/22
Model 3 Testing	2d	02/27/22	02/28/22
Sprint 7	**7d**	**03/01/22**	**03/07/22**
GUI Design	7d	03/01/22	03/07/22
Sprint 8	**7d**	**03/08/22**	**03/14/22**
GUI Implementation	7d	03/08/22	03/14/22
Sprint 9	**7d**	**03/15/22**	**03/21/22**
GUI Testing	7d	03/15/22	03/21/22
Final Report	7d	03/15/22	03/21/22

Image 7.4 Gannt Chart [26]

7.5 Gitlab Repositories



Valadez Mena, Rigoberto @rv21463 — 15 hours ago
Pushed to branch master
9143354e · Added placeholder text

Valadez Mena, Rigoberto @rv21463 — 16 hours ago
Pushed to branch master
67864c55 · Generated confusion matrices to the GUI

Valadez Mena, Rigoberto @rv21463 — 16 hours ago
Pushed to branch master
2ba9f715 · Updated preprocessing engine

Valadez Mena, Rigoberto @rv21463 — 17 hours ago
Pushed to branch master
9150771e · Loading images in the GUI
... and 1 more commit. Compare 24ab7e1f...9150771e

Ray, Dixa @dr21882 — 17 hours ago
Pushed to branch master
24ab7e1f · Merge branch 'master' of https://cseegit.essex.ac.uk/21-22-ce9x3/21...
... and 1 more commit. Compare 4860d79b...24ab7e1f

The number of merge requests merged by month.

Image 7.5 Gitlab Repositories

.

Bibliography

[1] N. Birbaumer, "Brain-computer-interface research: coming of age," Clin. Neurophysiol., vol. 117, no. 3, pp. 479–483, 2006.

[2] Sivakami, A. and Devi, S.S.," Analysis of EEG for motor imagery based classification of hand activities" in International Journal of Biomedical Engineering and Science, 2015, pp.11-22.

[3] G. Schalk and J. Mellinger, A practical guide to brain–computer interfacing with BCI2000: General-purpose software for brain-computer interface research, data acquisition, stimulus presentation, and brain monitoring, 2010th ed. Guildford, England: Springer, 2010.

[4] Bbci.de. [Online]. Available: https://www.bbci.de/competition/ii/Graz_description.doc.

[5] "BCI Competition II," Bbci.de. [Online]. Available: https://www.bbci.de/competition/ii/.

[6] A. Gramfort et al., "MEG and EEG data analysis with MNE-Python," Front. Neurosci., vol. 7, p. 267, 2013.

[7] Thalkar, S. and Upasani, D., "Various techniques for removal of power line interference from ECG signal" in International Journal of Scientific & Engineering Research, 2013, pp.12-23.

[8] "Repairing artifacts with ICA — MNE 1.1.dev0 documentation," Mne.tools. [Online]. Available: https://mne.tools/dev/auto_tutorials/preprocessing/40_artifact_correction_ica.html.

[9] Sciencedirect.com. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0042698900002352.

[10] Ucsd.edu. [Online]. Available: https://sccn.ucsd.edu/pipermail/eeglablist/2009/002696.html.

[11] S. Sun and J. Zhou, "A review of adaptive feature extraction and classification methods for EEG-based brain-computer interfaces," in 2014 International Joint Conference on Neural Networks (IJCNN), 2014, pp. 1746–1753.

[12] G. Dornhege, B. Blankertz, and G. Curio, "Speeding up classification of multi-channel brain-computer interfaces: common spatial patterns for slow cortical potentials," in First International IEEE EMBS Conference on Neural Engineering, 2003. Conference Proceedings, 2003, pp. 595–598.

[13] S. Sun, "The extreme energy ratio criterion for EEG feature extraction," in Artificial Neural Networks - ICANN 2008, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 919–928.

[14] I. Xygonakis, A. Athanasiou, N. Pandria, D. Kugiumtzis, and P. D. Bamidis, "Decoding motor imagery through Common Spatial Pattern filters at the EEG source space," Comput. Intell. Neurosci., vol. 2018, p. 7957408, 2018.

[15] Wikipedia contributors, "Cross-correlation," Wikipedia, The Free Encyclopedia, 16-Mar-2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Cross-correlation&oldid=1077554144.

[16] S. Labs, "Measuring entropy in the EEG - sapien labs," Sapien Labs | Neuroscience | Human Brain Diversity Project, 05-Feb-2018. [Online]. Available: https://sapienlabs.org/lab-talk/measuring-entropy-in-the-eeg/.

[17]“Quantile - statista definition,” Statista Encyclopedia. [Online]. Available: https://www.statista.com/statistics-glossary/definition/356/quantile/.
[18]“mne_features.feature_extraction.FeatureExtractor — mne_features 0.2 documentation,” Mne.tools. [Online]. Available: https://mne.tools/mne-features/generated/mne_features.feature_extraction.FeatureExtractor.html
[19]F. Farooq and P. Kidmose, “Random forest classification for p300 based brain computer interface applications,” in 21st European Signal Processing Conference (EUSIPCO 2013), 2013, pp. 1–5.
[20]M. Kaper, P. Meinicke, U. Grossekathoefer, T. Lingner, and H. Ritter, “BCI Competition 2003--Data set IIb: support vector machines for the P300 speller paradigm,” IEEE Trans. Biomed. Eng., vol. 51, no. 6, pp. 1073–1076, 2004.
[21]Rajesh Singla ; Haseena B.A 2013. BCI Based Wheelchair Control Using Steady State Visual Evoked Potentials and Support Vector Machines. [Online] Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.375.7704&rep=rep1&type=pdf
[22]S. Parui, A. K. Roshan Bajiya, D. Samanta, and N. Chakravorty, “Emotion Recognition from EEG Signal using XGBoost Algorithm,” in 2019 IEEE 16th India Council International Conference (INDICON), 2019, pp. 1–4.
[23]M. Rashid, N. Sulaiman, M. Mustafa, S. Khatun, and B. S. Bari, “The classification of EEG signal using different machine learning techniques for BCI application,” in Robot Intelligence Technology and Applications, Singapore: Springer Singapore, 2019, pp. 207–221.
[24]J. Liu, Y. Sheng, and H. Liu, “Corticomuscular coherence and its applications: A review,” Front. Hum. Neurosci., vol. 13, p. 100, 2019.
[25]Abhang, P.A., Gawali, B.W. and Mehrotra, S.C., 2016. Technological basics of EEG recording and operation of apparatus. Introduction to EEG-and speech-based emotion recognition, pp.19-50. P. A. Abhang, B. W. Gawali, and S. C. Mehrotra, “Technological basics of EEG recording and operation of apparatus” in *Introduction to EEG- and speech-based emotion recognition*. London, England: Elsevier Science, 2016. pp 51-79.
[26]Team 11, “Software Requirements Specification”, in Unpublished Manuscript, 2022, pp. 1-16.