UNIVERSITAT DE BARCELONA

Treball final de grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

NeuroClean: Multipurpose Neural Data Preprocessing Pipeline

Author: Manuel A. Hernández Alonso

Supervisors: Dr. Ignasi Cos, Mr. Michael DePass
At: Departament de Matemàtiques i Informàtica

Barcelona, 12 de juny de 2023

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

NeuroClean: Multipurpose Neural Data Preprocessing Pipeline

by Manuel A. Hernández Alonso

Electroencephalography (EEG) and Local field potentials (LFP) are two commonly used measures of electrical activity in the brain. These signals are used extensively in both industry and research and have many real world applications. Before any analyses can be performed on EEG/LFP, however, the data must first be cleaned. The main objective of this project was to create an unsupervised, multipurpose EEG/LFP preprocessing pipeline. Its unsupervised nature would, consequently, help alleviate problems involving reproducibility and biases that arise from human intervention. Moreover, manual signal cleaning is time and labor intensive. The adoption of an automated workflow would, therefore, save researchers valuable time and resources. A secondary goal was to allow the pipeline to be fit to several use cases, thus standardizing the cleaning methods used in neuroscience. We designed an automated EEG/LFP preprocessing pipeline, NeuroClean, which consists of five steps: bandpass filtering, line noise filtering, bad channel rejection, and independent component analysis with automatic component rejection based on a clustering algorithm. Machine learning classifiers were used to ensure task-relevant signals were preserved after each step of the cleaning process. We used an LFP dataset recorded from a *cynomolgus* macaque to validate the pipeline. Data was recorded while the monkey performed a reach-to-grasp task, and three sections of the movement were used for classification. NeuroClean appeared to remove several common types of artifacts from the signal. Moreover, it yielded over 97% accuracy (whereas chance-level is 33.3%) in an optimized Multinomial Logistic Regression model after cleaning the data, compared to the raw data which performed at 74% accuracy. The results show that NeuroClean is a promising pipeline and workflow that may be explored in the future. [1]

[1] Code available on GitHub

Acknowledgements

Firstly, I must thank one of my supervisors, Mr. Michael DePass, for his support and positive attitude towards the project. He guided me throughout the duration of the project and helping me whenever I needed it, being usually readily available. Secondly, I must thank Dr. Ignasi Cos for the opportunity of working in a really interesting and unexplored topic. He helped me whenever he had the chance to do it.

Contents

Chapter 1

Introduction

Local field potential data was collected from non-human primates by the Dancause Laboratory at the University of Montreal in Canada. This document describes a pipeline of automated EEG data preprocessing, whose performance is quantified by classification accuracy metrics. A secondary goal was to find possible parameters to fit a single workflow to multiple cases without changing the pipeline's steps. This section will start by reviewing the properties of neural data and what we mean by preprocessing. Secondly, we will distinguish between manual preprocessing and automated preprocessing. Subsequently, we will review how to assess the processed data. Finally, we will provide a high-level summary of the project.

1.1 Neural Data

Neural data, specifically electroencephalography (EEG) data, is a measure of electrical activity of the surface of the brain. EEG is a type of time series recorded in channels using electrodes **Figure 1.1** that are either attached to the scalp of the subject using conductive gels and adhesive materials, or surgically implanted. The result of these recordings is data in the form of voltage differences between the electrodes. Subsequently these differences are divided into channels, with one temporal series per channel. This data is usually recorded at high sampling rates to capture the sensitive and fast dynamic movement of brain activity. Local field potential (LFP) data is similar to EEG though it is normally recorded via implanted microelectrodes and, thus, has a higher spatial resolution. Consequently, LFP and EEG data are often subject to artifacts and noise captured by the electrodes when recording is performed.

1.2 Preprocessing of Neural Data

Neural data, and consequently, LFP and EEG data are affected by noise, from either physiological or non-physiological sources alike. Notably, it is most common to find line noise around 50Hz or 60Hz due to electrical contamination. Due to this, the raw data is

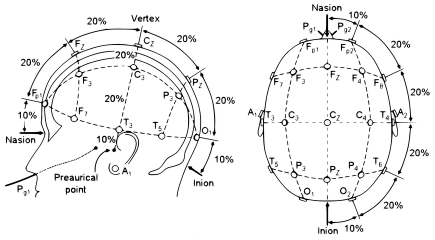


Figure 1.1: Diagram of the electrodes' placements

passed through a pipeline for denoising and artifact removal to obtain a clean signal of the brain dynamics.

This pipeline is designed to remove noise and artifacts step by step, maximising the signal preserved and the noise removed **Figure 1.2**. During these steps we can find different techniques like temporal series analysis, signal denoising and epoching.

It is important to emphasise that this step is often necessary to make performant models and, more generally, to perform any research involving brain activity data. Clean signals ensure the quality of the data and reduce the influences of noise. It is by this process that we can assure the robustness and reliability of the studies performed with EEG and LFP data. Nevertheless, the use of cleaning may result in worse performance of machine learning models.

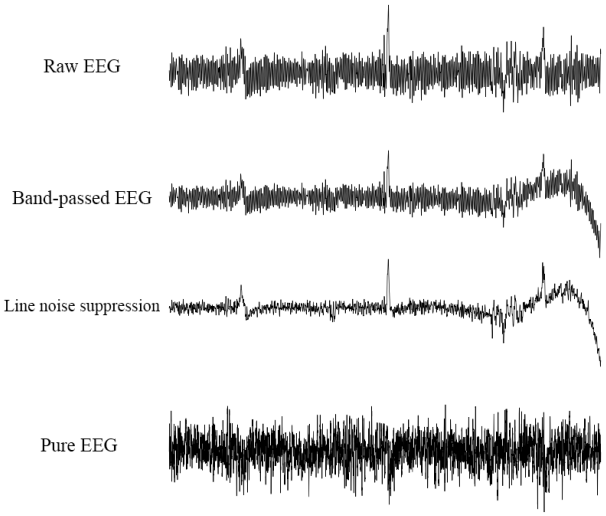


Figure 1.2: Signal on different stages of preprocessing

Preprocessing is performed by taking out artifacts from the signal, such as electrical noise at 50-60Hz, removing channels that are not functioning properly, performing independent component analysis (ICA) and rejecting the bad components, and other methods. The caveat is that most of these methods require some human intervention, such as choosing which components are artifacts and which components constitute real signal. It is by

this combination of computational tools and human supervision that we can get the cleanest data.

Nonetheless, requiring human intervention makes the pipeline susceptible to biases of the observers and consumes more time and resources as each component would need to be reviewed by a human. Consequently, results extracted from this human-machine cleaned data are not as consistent as an automated process. For these reasons, we would like to have a fully automated preprocessing pipeline that bypasses manual interaction with the data.

There have been several attempts to design automated preprocessing pipelines: The Harvard Automated Processing Pipeline for Electroencelography, Automatic classification of ICA components from infant EEG using MARA, etc.. However, some of these aren't fully automated, or they are case specific on account of being a hard problem to resolve adequately. Hence, the objective of this project is to create a fully automated preprocessing pipeline that can be fit to different use cases.

1.3 Preprocessing pipelines

As we have seen there are two types of pipelines: automated and supervised. For the purpose of this project we'll focus on automated pipelines, as these are the yet to be perfected ones. Furthermore, the use of fully automated pipelines comes with several benefits. Firstly, they ensure consistency and standardization across trials throughout the preprocessing process since they follow predefined algorithms and rules. Due to this, it also removes biases that come with human intervention making the data reliable and robust.

On another note, the use of automated pipelines maximizes flexibility and scalability; being able to process efficiently large and new EEG datasets without requiring manual adjustments, makes them better suited for large studies or clinical settings. Additionally, the use of these pipelines saves time, allowing researchers to focus their efforts on data analysis and interpretation rather than data cleaning and preparation.

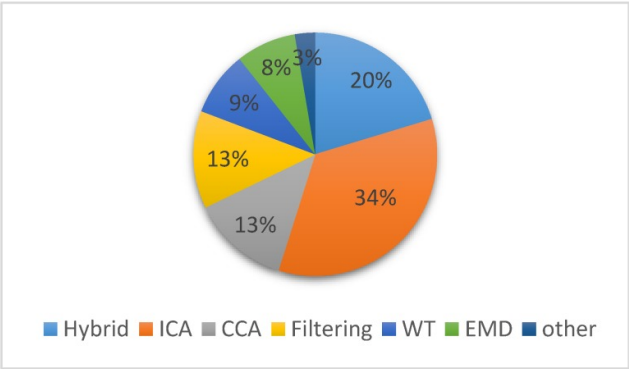


Figure 1.3: Percentage of preprocessing techniques: Percentage of the number of references published during the past three years (2016-2019), *(Jiang, et al., 2019)* [1]

While automated preprocessing pipelines provide these benefits, they are not yet intended to replace human expertise entirely due to the need to review data quality and validation. Experts typically assess the quality of the data by leveraging domain knowledge that may not be captured by the pipeline.

At the same time, preprocessing data comes with challenges that arise from segmenting good data from poor data. Some of the drawbacks of cleaning data are inadvertently removing important data or signal, and failing to remove noise or artifacts. Certainly, removing data from brainwaves may cause poor results when using the clean data due to missing key information to extract conclusions and results. Similarly, not successfully removing noise and artifacts may result in noise bias and poor reproducibility of research and investigations across experiments in different laboratories.

1.4 Neural Data Cleanliness Assessment

To minimize the amount of significant data removed from the signals we propose quantifying the quality of the data automatically with the use of classifiers to get several scores and statistical metrics to compare the raw and clean signal. These performance values give information that can be used to fit a single pipeline to multiple cases by adjusting parameters based on the information given by the classifier. Additionally, these statistical and performance values can be used to assess the quantity of noise/artifact removal in each step of the pipeline.

A multitude of classifiers can be used to assess the quality of the data including: logistic regression, K-nearest neighbors, support vector machines, tree-based methods, deep learning and others. Each classifier has its own advantages and disadvantages. Here we propose the use of multinomial logistic regression and K-nearest neighbors for the labeled data that we have. MLR was chosen due to it being a modest classifier regarding assumptions over the data. KNN on the other hand, was chosen due to being a non-linear solution based classifier that can outperform MLR if no linear solution can be found. Even if new data with new labels are used in the pipeline, swapping the utilized classifiers is supported as the performance values are not classifier dependant.

1.5 Pipeline and project proposal

Finally we'll give a high level summary of the proposed pipeline. Firstly, the goal of this particular pipeline is to make a general pipeline that can be fit to specific cases so that it can be unbiased and standardize across laboratories and trials, but self adjusted across research and experiments without manual intervention.

We segmented the project in 3 stages **(Figure 1.4)**: converting the data provided by the Dancause laboratory into standard LFP data of dimensions number of channels by number of samples with no labels; then we preprocess this standard LFP data by band-passing the signals, removing line noise, removing bad channels, performing ICA and a clustered version of MARA and epoching; finally the last stage is feature extraction (spectral amplitude in this case) and assessing signal quality via the performance metrics of two classifiers and statistical values of each of the extracted features.

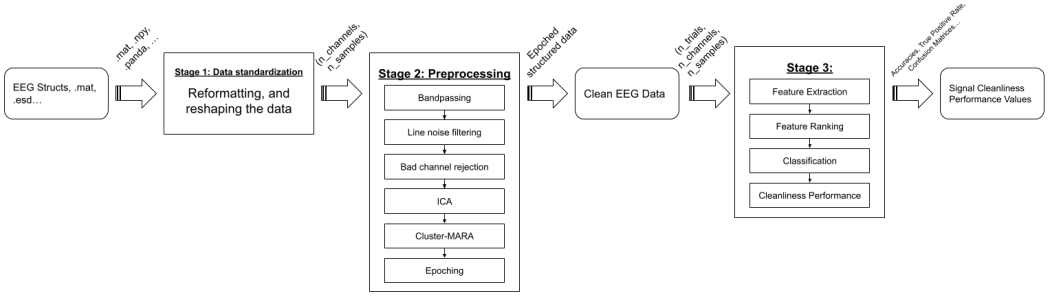


Figure 1.4: Project's proposed pipeline

The use of this pipeline may result in the loss of some trials or data, but statistically insignificant when we take into account the large amount of data used in research and clinical studies, thus making a compromise of some data for more cleanliness of the trials we end up using. Additionally, the use of a multi-purpose pipeline such as this one saves time by eliminating the manual adjustments across trials even if the execution of the pipeline requires some extra time.

In the observations and tests done through this project (progressive feature classification, shuffling labels, feature ranking, multiple and single class classification...), we noticed a better performance on 3 class classification through each step of the pipeline, suggesting good noise and artifact removal after the data was cleaned via the automated pipeline.

Chapter 2

Background

The idea of this section is to discuss current solutions for preprocessing pipelines and to quantify the cleanliness of signals, as well as to discuss different papers and studies relating to removing artifacts in the data. Additionally, we'll discuss the inspirations and precedents for the project.

2.1 Automated Preprocessing Pipelines

EEG preprocessing pipelines have started as simple algorithms that remove very specific artifacts (such as eye blinking, power supply noise, muscle spasms, etc.). Most of the pipelines use Independent Component Analysis (ICA) [4], or variations such as Wavelet Independent Component Analysis (wICA) [2], to reject artifactual components.

Many efforts have been made to create pipelines for EEG/LFP for artifact detection and removal, each one addressing different preprocessing aspects. The ADJUST pipeline (Mognon et al., 2011) [5] utilizes ICA to effectively remove stereotypical artifacts, notably eye blinks, eye movements, and discontinuities in the data. In another pipeline, HAPPE (Laurel J. et al., 2018) [2] the ICA is performed twice, once as wICA to soft remove some of the obvious artifacts (e.h., signal discontinuity) and then proper ICA with Multiple Artifact Rejection Algorithm (MARA), a machine-learning algorithm that evaluates the ICA components (Winkler I. et al, 2011) [6]. This HAPPE pipeline **(Figure 2.1)** was the main inspiration of this project's pipeline with changes of algorithms in each step and a multipurpose component rejection at the end.

Continuing on the MARA topic, most of these component rejection algorithms are based on pretrained classifiers that are fit using labeled data and predefined structures. Some efforts to perform automatic unsupervised component rejection algorithms have been performed using clustering such as the work done by Angel M. et al (2019) [11].

2.2 Case Specific Pipelines

The steps and parameters vary widely across studies with little standardization. For example, studies have been developed and tested on healthy adult EEG data with low

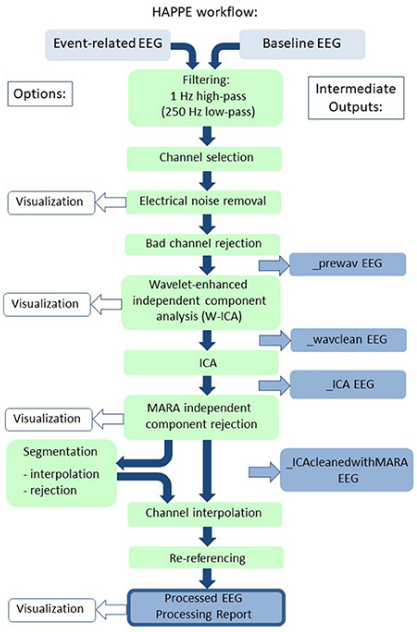


Figure 2.1: The Harvard Automated Processing Pipeline for Electroencephalography: Schematic illustrating the HAPPE pipeline's processing steps.

levels of artifact contamination (eg. Nolan, H. et al., 2010 [7]; Mognon et al., 2011 [5]; Bigdely-Shamlo et al., 2015 [8]; among others); but on the other hand there have been studies with infant EEG data (Laurel J. et al., 2018 [2]; Ana Fló. et al., 2022 [3]), where new challenges arise to extract and clean the proper brain signals. Finally, related closely to the data used and tested during this project, we have studies that use monkey LFP data (F. Sandhaeger. et al., 2019 [9]: Tomoya N. et al., 2020 [10]).

As more clinical applications and studies are done with EEG/LFP, a multipurpose framework or pipeline is needed to reduce costs and manual intervention, reducing biases and maximizing reproducibility, scalability and robustness when performing EEG related experiments.

2.3 EEG Data Quality Assessment

To score and rate the data, researchers from the Simon Fraser University, Canada, have looked into distinctive performance values and objective indexes to qualify the signal on its cleanliness (Shaun D. Fickling et al. 2019) [12], where a general purpose framework was set for the automated and objective assessment of the quality of EEG data. This framework uses six metrics that are shown to be highly sensitive to stereotypical artifacts, such as eye blinking, muscle interaction, etc.. A support vector machine classifier was used to identify the artifact contaminated EEG with 90.75% accuracy, 97.7% sensitivity, and 83.8% specificity. This finding by Shan D. et al. showed that objectively classifying artifact ridden EEG data is possible.

2.4 Unique Characteristics of the Present Project

This project is different compared to the aforementioned pipelines and studies in several key aspects. Firstly, the previous studies are mostly performed on low density EEG instead of the high density LFP data given by the Dancause Laboratory. The high density data is able to perform better when classifying and distinguishing between states of movement. Secondly, the previous studies and pipelines sought to solve and clean data for specific use cases. Instead, we are proposing a multipurpose framework that can fit itself to the given problem by discarding some extra data. Lastly, in the previously mentioned studies, no analysis of each step of the pipeline is done. They only reported performance metrics before and after the pipeline application. In the present work we quantified the effects of each step of the pipeline.

Chapter 3

Data and Methods

We'll begin by reviewing the experiment used in this document to create the preprocessing pipeline. The experiment was performed at the Dancause Laboratory of the University of Montreal. Then, we'll review the preprocessing pipeline in detail by explaining how the distinct steps were performed. Finally, we'll discuss the machine learning and statistical methods used to quantify the effect of the pipeline on the data.

3.1 Experiment: Reaching objects

3.1.1 Summary

Two macaque monkeys underwent a surgical procedure to implant four Utah multielectrode arrays, allowing the recording of 256 channels of local field potentials (LFPs) across the M1, PMv and PMd regions of the brain. Following the implantation, the monkeys were trained to perform a behavioral task that involved reaching for and grasping one of four grips on an experimental apparatus using a designated hand. Visual cues in the form of light signals indicated which hand and grip combination to execute. Successful completion of the task required grasping the designated grip with the designated hand which resulted in the delivery of a juice reward.

3.1.2 Behavioural Task

The macaques started by placing their hands on the home plate of the experimental apparatus. The presence of the hand was monitored via infrared sensors. After a variable resting time, two light cues were given to indicate which hand to use and which grip type to employ (either Precision or Power, four grips total). Each grip type required a different orientation of the arm and coordination of the hand to activate. Grasping the grips required a specific pronation of the forearm, the angle of this pronation could be manipulated by adjusting the grip orientation (0°, 45°, 90°, 135°). The GO signal for this task was the moment the light indicators turned off. The GO signal indicated subjects were allowed to grip the handles and obtain a drop of juice. Several sessions were recorded,

but for the purpose of this project we only focused on the trials of unimanual power grips with the right hand.

3.1.3 Neural Recordings

A set of chronic high-dimensional Utah electrode arrays were surgically implanted in the M1 area of the left hemisphere, as well as in the PMv and PMd regions of both hemispheres (a total of five arrays). Each array contained 96 electrodes. 256-channel LFP was sampled at 4882.81Hz and band-pass filtered between 100 and 2000Hz before being recorded digitally.

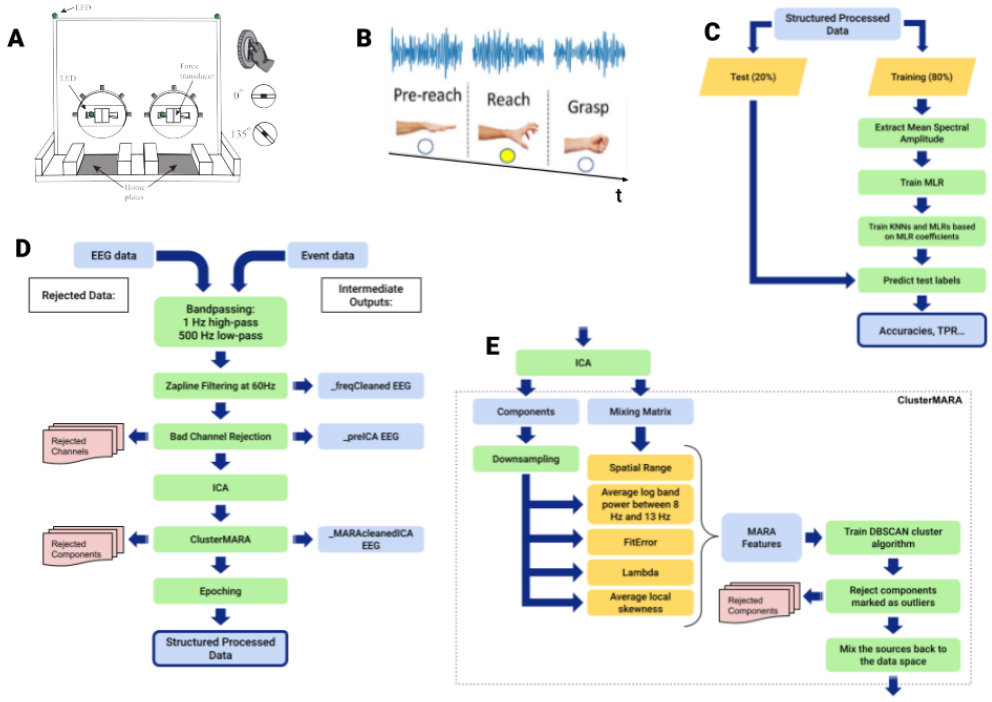


Figure 3.1: Experimental methods. *A*: Illustration of task setup, with home plates where the monkey had to position the hands to initiate the trial, and a schematic of the grip apparatus that the monkey had to reach for. *B*: Illustration of the three motor-related states to be classified with corresponding example and local field potential (LFP) signal above. *C*: Machine learning pipeline, the goal was to identify the classes based on the mean spectral amplitude of each channel. Eighty percent of the dataset is used to train the classifiers [m Multinomial Logistic Regressor (MLR) and m 1 Nearest Neighbours classifiers (1NN)], and twenty percent is used to test them. Finally, the pipeline gives the accuracies, true positive rates, and other performance metrics. *D*: Schematic of the preprocessing pipeline. It starts with a bandpass filter from 1Hz to 500Hz, followed by a zapline filter to remove power supply artifacts and their harmonics, then a bad channel rejection algorithm is applied, followed by an ICA with ClusterMARA to reject components, and finally the data is epoched to get a structured processed data. *E*: Summary of the MARA features and ClusterMARA. After the ICA the ClusterMARA computes 4 source components features after down sampling the components, and 1 pattern feature (Derived from the Mixing Matrix). With these features a DBSCAN cluster algorithm is fit. Following the DBSCAN, we reject the components marked as outliers and mix back the sources to the original data space.

3.2 NeuroClean: Proposed Preprocessing Pipeline

3.2.1 High-level Summary

NeuroClean is a multipurpose EEG preprocessing pipeline based on different algorithms to get to a final clean EEG signal separated by channels that can be traced back to the original electrodes. The several steps are as follows: a bandpass filter applied between 1Hz and 500 Hz to remove frequencies that interfere with EEG research and the following steps of the preprocessing pipeline, a Zapline (A. de Cheveigné, 2020, [13]) filter applied at 60 Hz to remove power supply noise and its harmonics, a bad channel rejection based on iterative standard deviation (Komosar, et al. 2022, [14]), an ICA performed to get components and remove them with a modified version of MARA (Winkler I., et al. 2011 [6]) based on clustering MARA features, and epoching based on events during the recording to get the different classes.

3.2.2 Data Standardization

Due to having multiple type of structures, matrices, and extensions for saving EEG and LFP data, we decided on using a standard matrix of number of channels by number of samples, with an extra meta data dictionary that contains information of the ratio of the sampling frequency of the data and the frequency given by the clock of the experiment (In case the frequency given by the clock is the same as the sampling frequency of the electrodes, the ratio will be 1). Lastly, an event file is added to epoch the data into several classes in which we can find the sample stamp of the trial events.

3.2.3 Pipeline

In this section, we'll review the specifics of each step of the pipeline **(Figure ??)**, as well as details regarding the input and output of each step. We'll also discuss the specific python module used on each step for reproducibility in case there is interest in reimplementation by other parties.

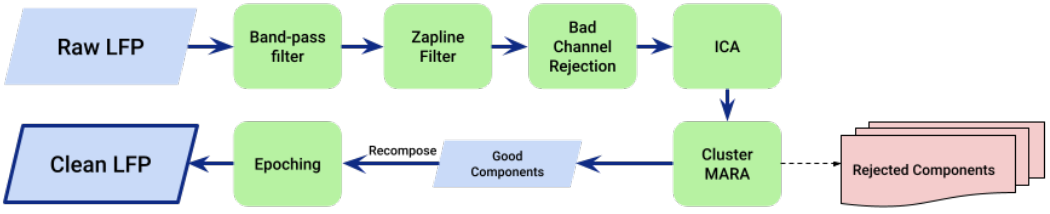


Figure 3.2: NeuroClean: Schematic of the Multipurpose Automated Preprocessing Pipeline.

Band-passing

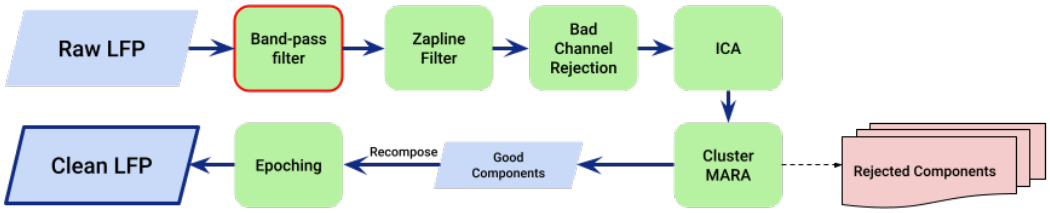


Figure 3.3: NeuroClean: Schematic of the Multipurpose Automated Preprocessing Pipeline. The marked step is band-passing at 1 Hz to 500 Hz.

Band-passing is used in this pipeline to reduce noise around the signal and focus on the brain wave data, that mostly occur in frequency bands around the alpha (8-15Hz), beta (15-30Hz) and low gamma (30-70Hz) bands. Research on the application of the butterworth band-pass on EEGs was made by S. S. Daud et al. 2015 [16], and it was found that the application of the butterworth band-pass reduces some of the noise produced by different sources, which means that it is a good preliminary step.

To apply the band-pass filter, an infinite impulse response filter was used. The module used comes from SciPy and follows the butterworth (Butterworth S. 1930, [15]) implementation. We used the 3rd order of the filter and applied it as a band-pass between 1 Hz and 500 Hz. The infinite impulse response (IIR) filter works by recursively applying past output values to the current input. This means that the IIR filter can be represented by a recursive equation involving the input, output, and coefficients of the signal.

Zapline Filter

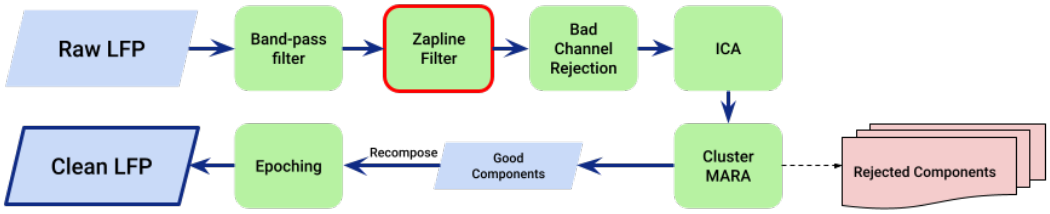


Figure 3.4: NeuroClean: Schematic of the Multipurpose Automated Preprocessing Pipeline. The marked step is the Zapline filter at 60 Hz.

The Zapline filter is used in this pipeline to remove power supply noise from the EEG data, for a better performance the data must be low-passed and high-passed before applying the Zapline filter. This method was developed by A. de Cheveigné (2019) [13] as a simple, yet effective way to remove power line noise and its harmonics from the EEG signal. Zapline combines the advantages of spectral and spatial filtering, while minimizing their downsides.

The algorithm is based on feeding the data to a 'perfect reconstruction filterbank' that separates the data **X** into two branches **X′** and **X″**. This first **X′** branch is assumed to be perfectly devoid of line artifacts as the separation method used is based on a square-shaped kernel κ of duration $1/f_{line}$, so the branch has zeros at the f_{line} and all its multiples. A second squared-shaped kernel is used, the complementary filter $\delta(0)$-κ, to extract the branch **X″**.

The next step of the algorithm is applying a denoising matrix **D** on the **X″** component. The denoising matrix is designed based on the JD/DSS method (A. de Cheveigné et al., 2014 [17]). More precisely, this method uses an artifact-enhancing bias filter β in the frequency domain that sets all Fourier coefficients that are multiple of f_line to one, and the rest to zeros. Then, a covariance matrix **M** is found with the branches and filter β and ordered by decreasing artifact power. Assuming that the first d components capture all the power line artifact, the denoising matrix **D** is obtained by getting last J-d columns of **M** by the last J-d rows of the pseudoinverse $\mathbf{M}^{+}$.

$$\tilde{\mathbf{X}} = \mathbf{X}' + \mathbf{X}''\mathbf{D} \tag{3.1}$$

Lastly, we mix back the two branches by adding them up as shown in the equation 3.1. The first term of the equation 3.1 is using a spectral filter to cancel the power line components, and the second term is using a spatial filter to cancel the rest. Since the Zapline method is based on a $1/f_{line}$ squared-shaped filter, the rest of the data that is not on the frequencies of f_line and its multiples is barely affected.

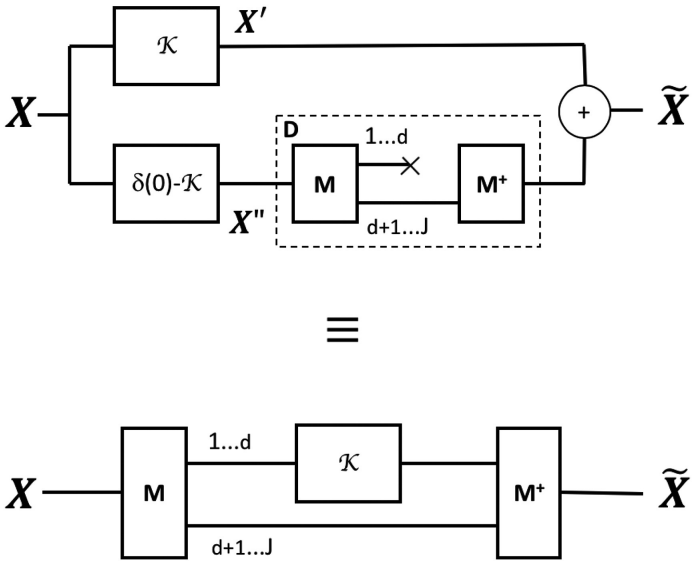


Figure 3.5: **ZapLine: A simple and effective method to remove power line artifacts**: *Top*: the signal is split into two branches, one that is spectrally filtered to remove line artifact (upper), the other that is spatially filtered to remove line artifact (lower), then the two branches are combined. *Bottom*: swapping linear operations (κ and **M**) leads to an equivalent pipeline where it is obvious that d dimensions are spectrally filtered, while the rest are intact.

To implement this method on this pipeline we used the dss_line function on the 60Hz frequency from the meegkit that implements the previous discussed algorithm. This meegkit[18] is a collection of EEG and MEG denoising techniques for **Python 3.8+**. Meegkit is hosted on Github and is open source.

Bad Channel Rejection

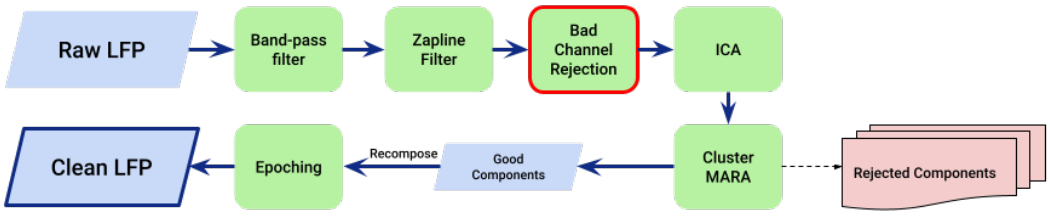


Figure 3.6: NeuroClean: Schematic of the Multipurpose Automated Preprocessing Pipeline. The marked step is the Bad Channel Rejection based on Kosomar, et al. 2022 [14].

Bad channel rejection was used in this pipeline to remove channels derived from broken electrodes, artifact ridden channels, or otherwise unusable channels from the data. The method used for this step is based on Kosomar, et al. 2022 [14] automatic iterative standard deviation bad channel detection method.

The algorithm starts by calculating the standard deviation of the signal for the j-th channel over the whole sample size of length N, equation 3.2.

$$\mathbf{SD_j} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left|V_{(i,j)} - \overline{V}_j\right|^2} \tag{3.2}$$

Where V_i is the i-th sample out of the N samples for the j-th channel and $\overline{V}_j$ is the mean voltage for the j-th channel.

Then, five criteria are established to detect bad channels, equations 3.3 to 3.7. The criteria used to reject the channels are as follows: the median and the 75th percentile range, the standard deviation of channels lower than $10^{-1}\mu$V and higher than 100μV.

$$\left|\mathbf{SD_{(j,k)}} - \mathbf{M_k}\right| > \mathbf{75th\ percentile} \tag{3.3}$$

$$\mathbf{SD_{(j,k)}} < 10^{-1}\mu V \tag{3.4}$$

$$\mathbf{SD_{(j,k)}} > 100\mu V \tag{3.5}$$

Two additional criteria for ending the iterations are:

$$\mathbf{SD_{p(k)}} > 5 \tag{3.6}$$

$$\mathbf{C_{r(k-1)}} = 0 \tag{3.7}$$

Where $SD_{(j,k)}$ is the standard deviation of the j-th channel at the k-th iteration. M is the median of the standard deviations of the channels at the k-th iteration. SD_p is the standard deviation of all individual channel standard variation in the k-th iteration. $C_{r(k-1)}$ is the number of bad channels detected in the previous iteration. Note that this last criterion is not present in Kosomar, et al. 2022 [14].

We implement this directly into the pipeline using standard NumPy methods and functions. If an individual channel, or multiple channels, is detected as a bad channel by the criteria of this method, it will be removed completely from further use in the pipeline. Additionally, we preserve a back mapping of the remaining channels to the original channels for analysis and research after the pipeline is completed.

ICA and Cluster-MARA

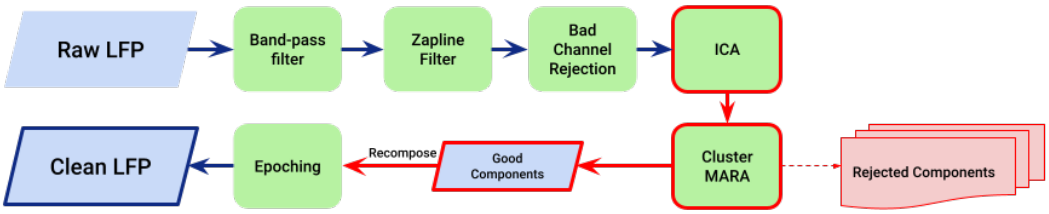


Figure 3.7: NeuroClean: Schematic of the Multipurpose Automated Preprocessing Pipeline. The marked steps are Independent Component Analysis (A.Hyvarinen, et al. 2000 [19]) and Cluster MARA, a modification on the algorithm presented in Winkler I., et al. 2011 [6].

This step is separated in two parts, we first apply the fast Independent Component Analysis (FastICA in A. Hyvarinen, et al. 2000, [19]) to extract as many components s as channels c are remaining; then we apply a modification of the MARA algorithm proposed by Winkler I., et al. 2011 [6] using clustering instead of pretrained models. This strategy combines the advantages of being a powerful supervised feature based component rejection algorithm, and being a clustering method that can be fitted to any EEG cleaning case.

Starting with ICA, we'll look at how the ICA is applied by the module used in the pipeline, specificity will be lost as this document is not focused on explaining all the inner workings of ICA (See A. Hyvarinen, et al. 2000 [19] for further information). The goal of ICA is to find the latent statistical variables. To achieve this, we first assume that there is are n linear mixtures $x_1, \cdots, x_n$ of statistically independent components s_k, these components are random variables.

It is convenient to use matrix notation instead of sum notation like shown before, so we denote $\mathbf{A}$ as the matrix of elements a_ij that represent the matrix for mixing and demixing the data from the original data space $\mathbf{x}$ to the component space $\mathbf{s}$. All vectors

are understood as column vectors, meaning that $\mathbf{x}^{\mathbf{T}}$, or the transpose of $\mathbf{x}$, is a row vector. Consequently we get the ICA statistical model equation:

$$\mathbf{x} = \mathbf{A}\mathbf{s} \tag{3.8}$$

ICA is a generative model, meaning that it describes how the given data is generated by a process of mixing the latent components s_i. Latent components means that the variables cannot be directly observed. Additionally, we assume that $\mathbf{A}$ is unknown. Finally, this means that we must estimate both $\mathbf{A}$ and $\mathbf{s}$ using the observable data $\mathbf{x}$.

Since the estimation is done over $\mathbf{A}$ and $\mathbf{s}$, we'll rewrite the equation to compute the inverse of $\mathbf{A}$, say $\mathbf{W}$ or weights matrix, and then obtain the independent components $\mathbf{s}$ by simply applying:

$$\mathbf{s} = \mathbf{W}\mathbf{x} \tag{3.9}$$

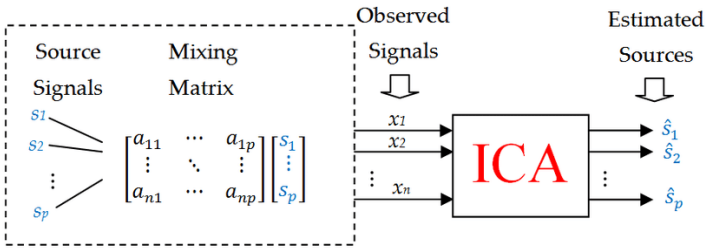


Figure 3.8: ICA schematic summary by D. Mika, et al. 2019 [20]

Now the specific way to estimate the $\mathbf{W}$ matrix is different depending on the algorithm used. In this project, we used the FastICA algorithm. The algorithm starts by centering the data $\mathbf{x}$ to make sure that $\mathbf{x}$ is a zero-mean variable. This is achieved by subtracting its mean vector $\mathbf{m} = E\{\mathbf{x}\}$.

Then we whiten the data before applying the ICA algorithm. This means that we transform the data vector $\mathbf{x}$ to a new vector $\tilde{\mathbf{x}}$ which is white, that is to say, the components are uncorrelated and their variances equal unity. Consequently, a white vector has a covariance matrix that equals the identity matrix:

$$E\left\{\tilde{\mathbf{x}}\tilde{\mathbf{x}}^T\right\} = \mathbf{1} \tag{3.10}$$

After the whitening, the actual ICA starts, the basic form of the FastICA algorithm is as follows:

1. Choose an initial random weight vector $\mathbf{w}$.
2. Let $\mathbf{w}^+ = E\{\mathbf{x}g(\mathbf{w}^T\mathbf{x})\} - E\{\mathbf{x}g'(\mathbf{w}^T\mathbf{x})\}\mathbf{w}$.
3. Let $\mathbf{w} = \mathbf{w}^+ / \|\mathbf{w}^+\|$.
4. If not converged, go back to 2.

Nevertheless, these steps are to get one unit of the independent components s_i. To achieve the weight matrix W we need to apply the one-unit method previously seen across several units with weight vectors $\mathbf{w_1}, \cdots, \mathbf{w_n}$. To avoid getting to the same maxima on more than one weight vector we have to decorrelate the outputs given by the vector-wise equation 3.9 as $\mathbf{s_i} = \mathbf{w_i}^T \mathbf{x_i}$ after every iteration. The way of achieving this decorrelation is by using a deflation scheme based on a Gram-Schmidt-like decorrelation, or by using a symmetric decorrelation that show no vectors privileged over others.

After applying the FastICA, we are left with 2 new distinct outputs the weight matrix $\tilde{\mathbf{W}}$ and the components $\mathbf{s}$. Note that the actual result of the FastICA contains an extra matrix that whitens the data when mixing and demixing, this will be implied with the use of $\tilde{\mathbf{W}}$ instead of $\mathbf{W}$. These two outputs are the base of the features used in MARA.

Multiple Artifact Rejection Algorithm (MARA) is an automatic method for classification of general artifactual source components (Winkler I., et al. 2011, [6]). This method uses six features derived from the component's time series, the component spectrum, the component's pattern and the component's weight vector. After the feature extraction, MARA uses supervised classification to distinguish between good and bad components. Consequently, MARA needs prelabeled data to properly reject artifactual components. Additionally, one of the six features (current density norm) that the original MARA uses requires spatial data relating to patterns of brain activity, which we didn't have for this project, so we discarded this feature.

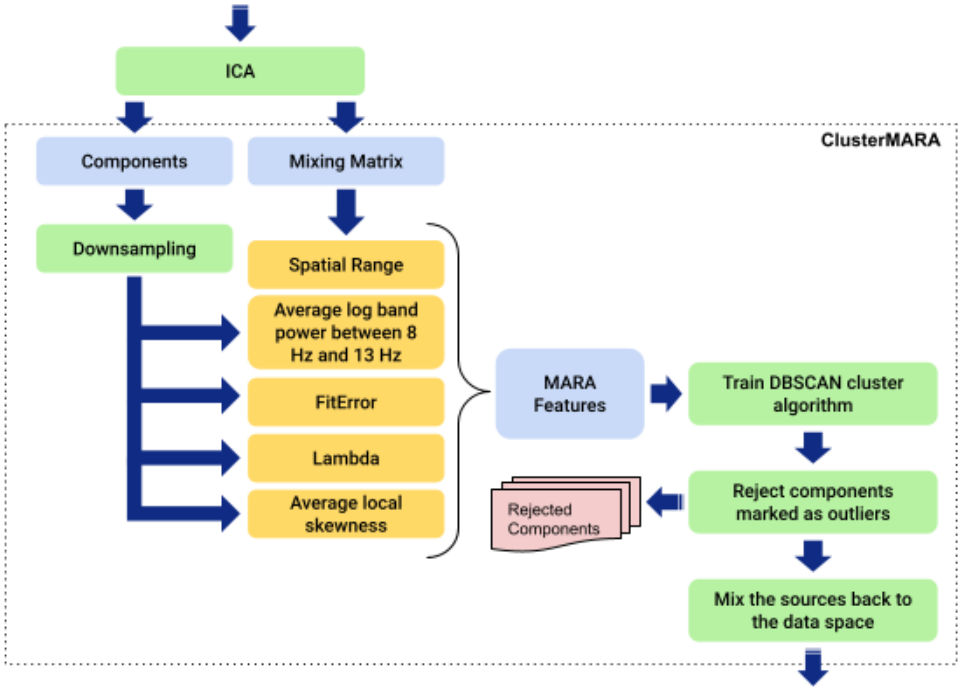


Figure 3.9: A summary schematic of the ClusterMARA algorithm, a modification on the MARA algorithm presented by Winkler I., et al. 2011 [6]

To avoid this issue we assumed that brain waves from the same brain area must have similar features that are clustered in one or several spaces in the MARA feature space. Using this assumption, we utilized a DBSCAN clustering algorithm to group the MARA features of the components **s**, getting clusters and outliers. We called the use of this method ClusterMARA **(Figure 3.9)**.

Firstly, we computed the following features for each component s_i:

- **Spatial range within pattern**, defined as the logarithm of the difference between the maximal and minimal activation of the weight vector.
- **Average log band power between 8 and 13 Hz**, defined as the average log band power of the alpha band between 8 Hz and 13 Hz. The power is computed using Welch's method for estimating the power spectral density of the signal.
- **lambda** λ, describes the variance parameter of the fitting of the component's power spectral density to the prototypical $1/\text{frequency}$ curve observed in brain waves.
- **Fit Error**, similar to λ, this feature describes the error in fitting of the component's power spectral density to the prototypical $1/\text{frequency}$ curve.
- **Average local skewness in 15 second windows**, as the name suggests, a sliding window of 15 seconds is applied over the component's time series where the skewness is computed, then every computation is averaged to get the mean absolute local skewness of the component. Skewness is defined as the measure of asymmetry of the distribution of the series.

Secondly, after these features are computed we are left with a matrix of s components by 5 features, meaning that we now have to find clusters in this 5-D space. To achieve this, we apply Density-Based Spatial Clustering of Applications with Noise (DBSCAN, Ester M. et al. 1996 [21]) to the matrix, this finds the core samples of high density and expands clusters from them. In this DBSCAN method, a parameter representing the maximum inter-sample distance to be considered part of the same neighborhood was tweaked until getting a value of 2; another parameter relating to the minimum number of samples in a neighborhood was set to 2 as well.

Then each component was marked and labeled with different cluster labels, or as outliers. For the purpose of this project we only rejected the outliers, but there exists an interesting exercise and expansion possibility of iterating over the combination of clusters and rejecting each combination. This is due to the assumption that artifactual components will be only outliers, but it could be the case that there exist entire clusters of artifactual components. Finally after rejecting the components by setting that component time series to zeroes, we mix back the data using the mixing matrix given by the FastICA algorithm.

We must remark that this ICA and MARA process was done per brain areas (M1, PMvL, PMvR, PMdL...), as per the assumption made in ClusterMARA that the components are only clustered if they come from the same brain area.

Finally, we implemented this fully two-part step by using the FastICA function in ScikitLearn that was implemented following A. Hyvarinen, et al. 2000 [19]; for the MARA feature extraction it was performed directly in the script using functions from SciPy and

NumPy, and finally for the clustering with DBSCAN, the function from ScikitLearn was used, that was also implemented following Ester M. et al. 1996 [21].

Epoching

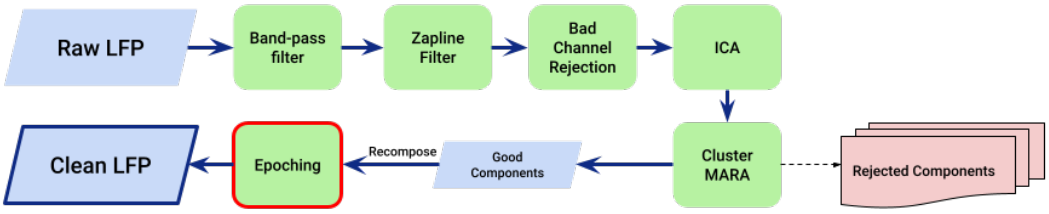


Figure 3.10: NeuroClean: Schematic of the Multipurpose Automated Preprocessing Pipeline. The marked step is the final step, epoching.

Finally, epoching is required for classification methods and to remove all parts of the time series representing downtime between trials. This step is also used to standardize the length of classes and to output a fully structured data of size n trials by m classes by c channels by j samples. The classes **Figure 3.11** used in this project were prereach, reach and grasp due to the inconsistency of the other classes in the given data.

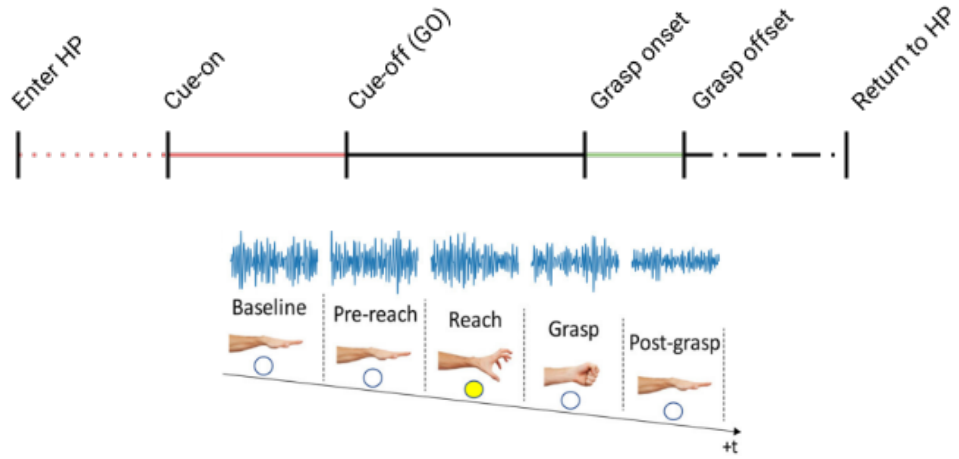


Figure 3.11: Classes and epochs given by the timestamps and events.

The method used for epoching is performed with timestamps given by the event data that the pipeline uses. Every class must be at least 500 time-points/samples; if a class doesn't meet this requirement, the specific trial is rejected from the group. After checking the duration of the classes in the trial, the classes are then centered around the midpoint

of each class and 500 samples are taken, 250 to each side. The data is then saved and exported.

The implementation of this step was done with pure python code with some NumPy methods used to speed up the process.

3.3 Metrics

To assess the cleanliness of the signal we used a machine learning approach where the goal of the classifier was to identify the different classes defined previously.

3.3.1 Machine Learning Pipeline

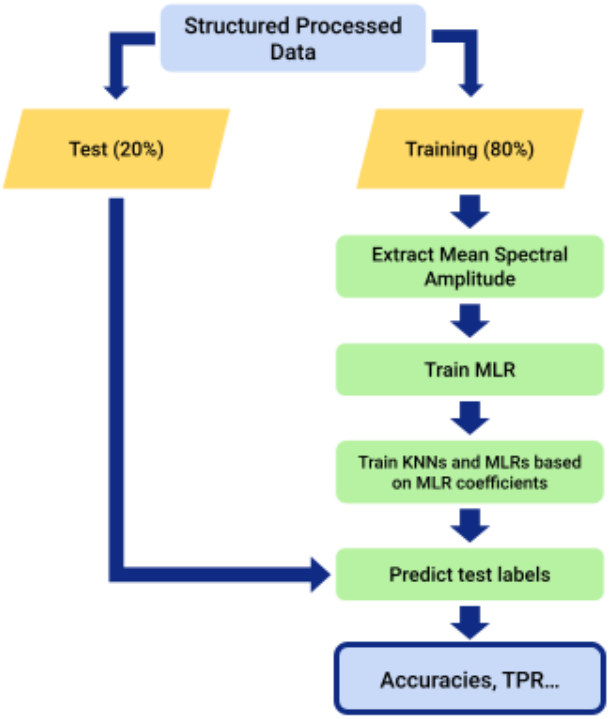


Figure 3.12: Schematic of the Machine Learning approach to review and assess the cleanliness of the signal

A machine learning pipeline was used over the structured preprocessed data given by the NeuroClean pipeline. This step was required in order to assess the cleanliness of the signal.

The pipeline starts by segmenting the data in train and test split of 80% training and 20% testing. Subsequently, the mean spectral amplitude $\overline{\mathbf{A}}$ is calculated for each channel of the data:

$$\overline{\mathbf{A_i}} = \frac{\sum_{j=1}^{N} |V_{i,j}|}{N} \tag{3.11}$$

Where $\overline{\mathbf{A_i}}$ is the mean spectral amplitude for the i-th channel, $V_{i,j}$ is the voltage at the j-th sample in the i-th channel, and N is total number of samples. With this we end up with a vector matrix of features for each class sample. The previous step before processing is leveling the data such that we have x number of class samples and y labels corresponding to each class sample.

Then the data is used to train a Multinomial Logistic Regression (MLR) model. This trained model then has trained coefficients of the features in the decision function. By ordering the trained coefficients in descending order, we get a variable that shows the importance of each feature, since the closer the feature is to zero, the less effect the feature has over the decision function. With this in mind, the indices of the coefficients are saved to use as the indices of the most important features.

Using this ordered indices list, we train four models iteratively, two 1-Nearest Neighbors (1NN) model and two MLR. One of each kind of model is assigned to be trained with shuffled labels to determine if the results of the shuffled data coincides with the expected theoretical random guessing model accuracy, i.e the model is trained with 3 classes, then the expected accuracy for the model should be approximately 33%. The remaining models are then trained with the first d features from the data X until we get to the total number of features D. Additionally, we do an r number of repetitions to fit the model several times to avoid random chance accuracies.

The goal of this step is to get the accuracies for each number d of features to determine the optimal number of features. The iteration works as follows:

1. Let $d = 0$.
2. Let d be the number of features for the current iteration.
3. Let X_d be the data with only the number of features d.
4. Fit a MLR and a 1NN model with the X_d data and the y class labels.
5. Let $a_{(d,r)}$ be the accuracy of the models at the number of features d and repetition r, calculated by predicting the class labels for the training data $X_{t,d}$ with the d features.
6. Let y_s be a random permutation of y.
7. Fit a MLR and a 1NN model with the X_d data and the s_y class labels.
8. Let $a_{s(d,r)}$ be the accuracy of the shuffled models at the number of features d and repetition r.
9. If the repetition r is less than the number of repetitions R, return to 4.
10. Let $\overline{a}_d$ and $\overline{a}_s(d)$ be the mean accuracies of the models at the number of features d, calculated by predicting the class labels for the training data $X_{t,d}$ with the d features.

11. If the number of features d is less than the number of features D, then increment d and return to 2.

Additionally, we implemented a tolerance and epsilon method that will stop the iteration early if the improvement in the last 30 features hasn't improved the accuracy by at least an ϵ number, i.e $10^{-}3$.

3.3.2 Accuracy and Confusion Matrix

The preceding pipeline outputs a set of accuracies and confusion matrices that represent how the models performed on each iteration and number of features. Since the goal of the NeuroClean pipeline is to clean the data, we were interested in the hit rate, or accuracy, of the models, meaning the predictions where the predicted class label and the actual class label matched.

In a confusion matrix, where the columns represent the real class labels and the rows the predicted class labels, the accuracy is calculated using the sum of diagonal of the matrix divided by the sum of the entire matrix.

$$Accuracy = \frac{Number\,of\,Correct\,Predictions}{Total\,Number\,of\,Predictions} \tag{3.12}$$

A possible problem that may arise from using accuracy is data imbalance. An imbalance of the data may lead to a high accuracy score, but internally, the class is misclassifying the minority classes and properly classifying the majority class. In other words, the model would be behave similarly to predicting always the majority class.

This problem does not exist in this project as the data used is balanced, with the class samples divided in thirds and distributed to each class label: Prereach, Reach and Grasp. Consequently, we used the accuracy to measure the cleanliness of signal and the effect of the pipeline on data classification.

3.3.3 Other statistical values

Apart from assessing cleanliness, we estimated the statistical significance of each feature in the data and mapped it back to the original electrodes. This may be interesting for further researchers to decide whether NeuroClean is a pipeline that fits its use case. Finally, as an extra step, we mapped the MARA features on a PCA plane to revise how many components are needed to explain 95% of the variance of the data, to confirm if the ClusterMARA algorithm is a good approach.

Chapter 4

Results

In this section we will review the results of the pipeline effect on the data and on the machine learning pipeline. Additionally we'll review the results of the clustering approach of MARA.

4.1 NeuroClean's Effect on Signals

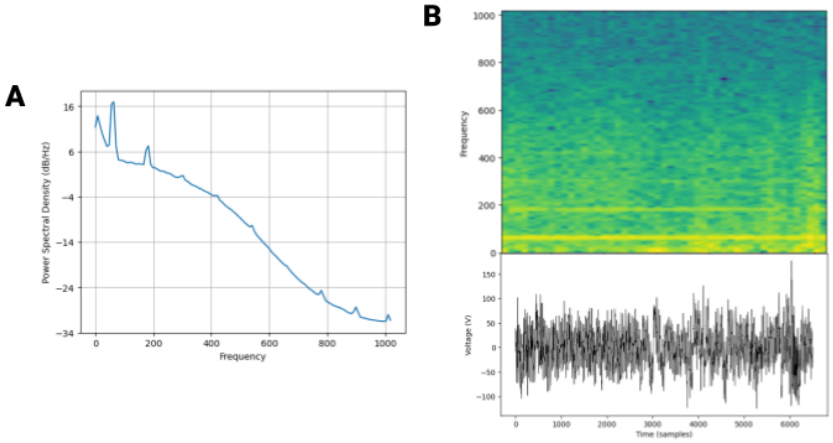


Figure 4.1: *A*: Power Spectral Density of the 12th channel of the PMdR brain area from the raw data. *B*: Spectogram and time series of the 12th channel of the PMdR brain area from the raw data

NeuroClean uses the previously mentioned steps to clean the EEG. After each step, some latent or observable artifacts are removed. The data used was 256-channel LFP data collected from the M1, PMv, and PMd brain areas. The pipeline processed the whole duration (approximately an hour) of the experiment performed at the Dancause Laboratory. Note that epoching and bad channel rejection don't have any effect on the signal as they

work by rejecting sections of data and not by transforming the signal.

We'll review the signal on a significant channel **Figure 4.1** where the effect of the pipeline is clear. This would be the 12th channel of the PMdR area during the 277th trial of the data (You can find the effect of the pipeline on the different brain areas together in appendix A).

Band-passing

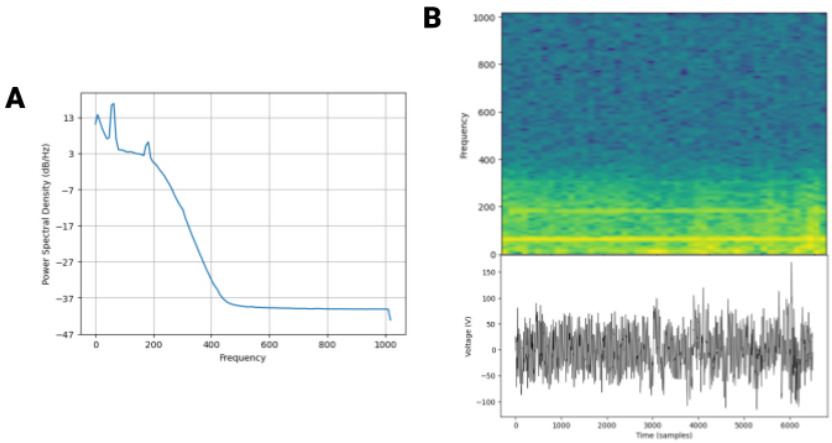


Figure 4.2: *A*: Power Spectral Density of the 12th channel of the PMdR brain area from the data after being band-passed. *B*: Spectogram and time series of the 12th channel of the PMdR brain area from the data after being band-passed

The effect of this step on the signal is clear, the frequencies after 500Hz have been filtered out and left at a power of -47dB as seen in **Figure 4.2-A**, which is effectively zero. In the time series seen in **Figure 4.2-B**, the line has thinned out as the high end frequencies have been filtered out, and just the ones up to 500Hz have been left in the mix. Finally, the general power of the frequencies have been lowered by around 3dB: seen by comparing the scales of **Figure 4.1-A** and **Figure 4.2-A**, and the general color in the spectograms in **Figure 4.1-B** and **Figure 4.2-B**.

Zapline Filter

The effect of this step on the signal is distinctively noticed in the reduction of the power of the frequency spikes at 60Hz and its harmonics (120Hz and 180Hz), see **Figure 4.3-A**. The line that was seen at around 180 Hz in the spectogram in **Figure 4.2-A** has been partially cleared.

But the main effect can be noticed in the time series in **Figure 4.3-B**, as the form and shape of the wave can now be seen more clearly without the noise and extra waves that were seen on previous figures. These two properties shows that the Zapline filter is properly removing power line noise from the signal.

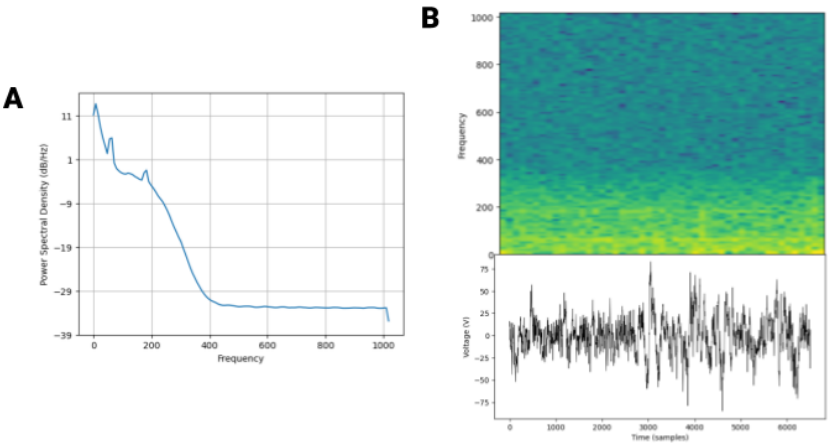


Figure 4.3: *A*: Power Spectral Density of the 12th channel of the PMdR brain area from the data after being band-passed, and zapline filtered. *B*: Spectogram and time series of the 12th channel of the PMdR brain area from the data after being band-passed, and zapline filtered

ClusterMARA

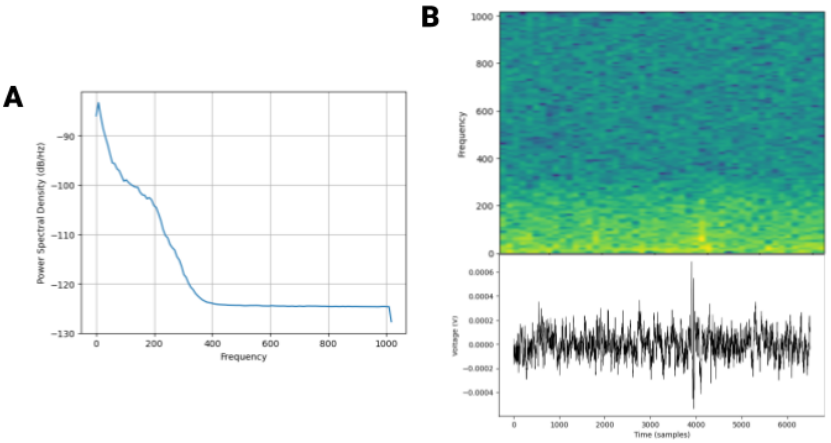


Figure 4.4: *A*: Power Spectral Density of the 12th channel of the PMdR brain area from the data after being band-passed, zapline filtered, and passed through ClusterMARA. *B*: Spectogram and time series of the 12th channel of the PMdR brain area from the data after being band-passed, zapline filtered, and passed through ClusterMARA

Here we notice how the power supply artifacts have been removed completely, the power at 60 Hz, 120 Hz and 180 Hz have completely disappeared from the power spectral density graph in **Figure 4.4-A**, and the line in the 180 Hz seen in the spectogram of **Figure**

4.4-B is also gone. This means that the ClusterMARA algorithm is properly removing the rest of the artifacts that the previous two signal cleaning methods couldn't remove.

The time series is also different, but it corresponds to the 12th electrode in the original data. Some back mapping have to be performed to translate the remaining channels to the original ones after the bad channel rejection step.

4.2 ClusterMARA

Previously we saw a modification of the MARA algorithm that we called ClusterMARA. This modified algorithm based its features on the original MARA features, except one. With the use of a DBSCAN model, we managed to cluster the different features into several clusters. The original hypothesis is that brainwaves must be close to each other in the MARA feature space.

The data used for the clusterMARA was the output provided by the FastICA algorithm in the SciKit Learn python module. This FastICA algorithm provided the mixing matrix (That already contained whitened coefficients) needed to transform the data to the source space. The FastICA algorithm was applied per brain area, PMvR, M1, PMdR, and PMdL; the clusterMARA algorithm was also applied per brain area. To visualize the results a

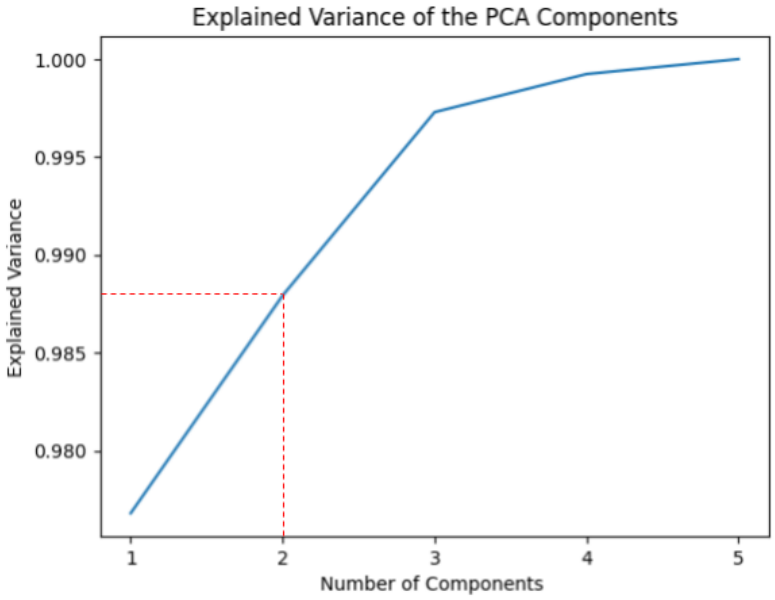


Figure 4.5: Plot of the accumulated explained variance by the number of components N

Principal Components Analysis was used to transform the 5 dimensional features to a lower N dimensional principal component space. The number of principal components used to describe the data was 2 due to them having an accumulated explained variance of 98,7% **Figure 4.5**.

After the PCA was applied we plotted the PMvR data as an example into a scatter plot with a color assigned to each one of the clusters found in the DBSCAN model (See brain areas appendix for the clusters of the rest of the brain areas):

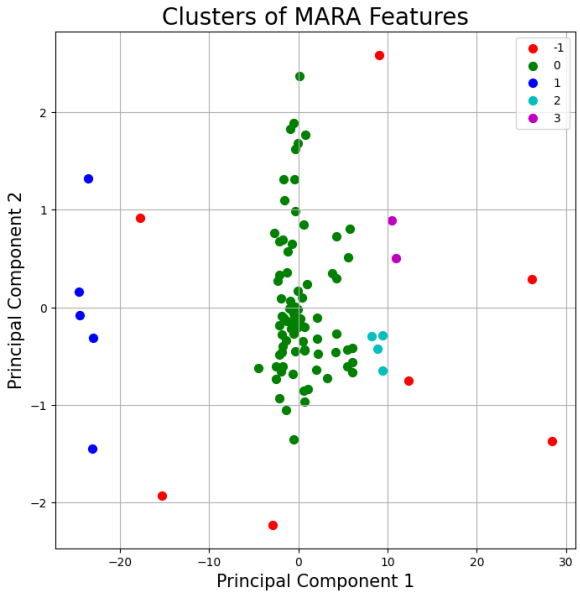


Figure 4.6: Scatter plot of the different clusters found in the DBSCAN model. The data was projected to 2 principal components that explain 98,7% of the variance

Here the labels 0 to 3 are the clusters of data, the label -1 is reserved for the outliers that don't fit into the clusters, these were the rejected components. Addressing the Cluster-MARA initial hypothesis, we see that most of the data lives in a big cluster in the middle in green, with a few really close clusters in cyan and magenta, this seems to indicate that the brain waves from the same brain area are clustered together. But as we can see there is a cluster that stands out because it is a cluster that is further away from the other clusters, the cluster 1 in blue. To review what each cluster of data represented, we took the first few ICA components of each cluster label and plotted a section of the time series in a matching color to the clusters labels found in **Figure 4.6**.

In **Figure 4.7** we see that the clusters 0, 2 and 3 seem to have similar data that can be interpreted as brain wave data. But the cluster 1 has clear artifactual data possibly related to muscle contractions, eye blinking, or other similar artifacts, since the time series seem to have periodic spikes not prototypical of brain activity. Finally, the outliers seem to be related to power supply noise, and other related interference, due to these being ones that we rejected. Finally, in a previous figure (**4.4**), we saw that the power lines at 60 Hz, 120 Hz and 180 Hz disappeared from the channels after applying ClusterMARA, this is why we think that the outliers clusters in **Figure 4.7** are probably related to power supply noise, and other related interference due to these outliers being rejected in the ClusterMARA step of the pipeline.

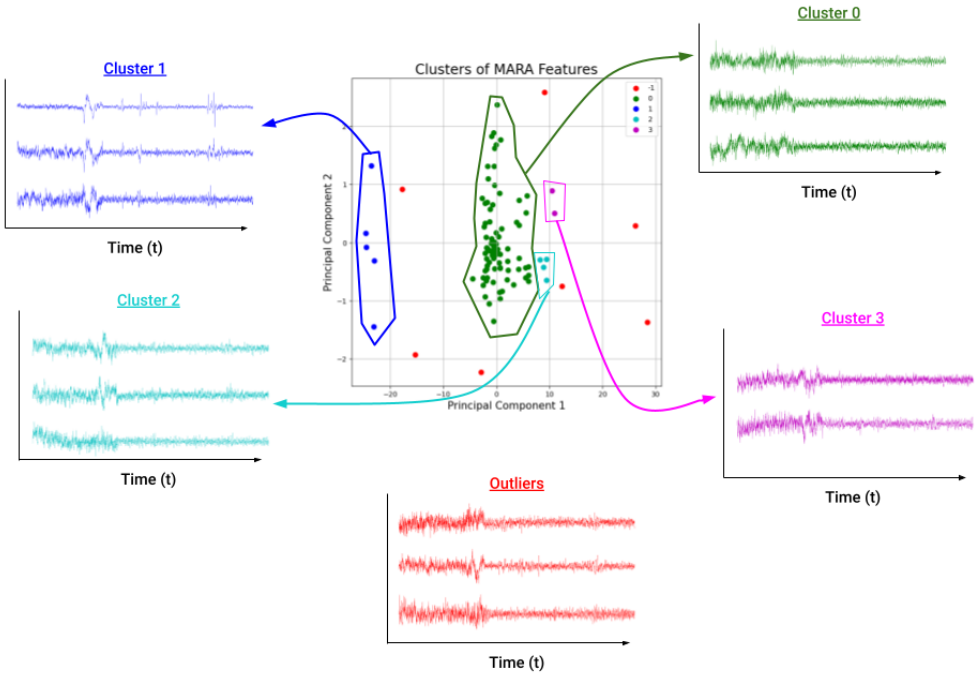


Figure 4.7: Scatter plot of the different clusters found in the DBSCAN model. The data was projected to 2 principal components that explain 98,7% of the variance

4.3 Machine Learning Pipeline

To assess the data cleanliness a machine learning classification pipeline was designed and used. This ML pipeline used as input the data outputted by NeuroClean, a n_trials by $n_classes$ by $n_channels$ by $n_samples$ multidimensional matrix. We'll now discuss the effect the distinct NeuroClean's steps had on the classification power given by the behavioural task performed by the Dancause laboratory.

The data collected after each step was epoched into the previously mentioned multi-dimensional matrix. Here the analyzed statistical and performance metrics were overall model accuracies and confusion matrices. The statistical significance of the Multinomial Logistic Regression was also analysed and backmapped to the original E electrodes.

With the machine learning pipeline an iterative method was defined to get the accuracies across all number of features, from only one feature used to N number of features. After this, an optimal number of features was defined using a tolerance and epsilon. This optimal number of features gave us an optimised model for each step of the pipeline. All graphs shown that include separation by steps are cumulative graphs, meaning, when a step is mentioned, all previous steps are included, i.e Bad Channel Rejection step includes the previous Band-passing step and ZapLine Filter step. A possible grid search was initially planned, but some of the steps required previous steps to function properly and/or required massive amount of time, so the idea was discarded.

4.3.1 NeuroClean's Steps' Effect on Classification

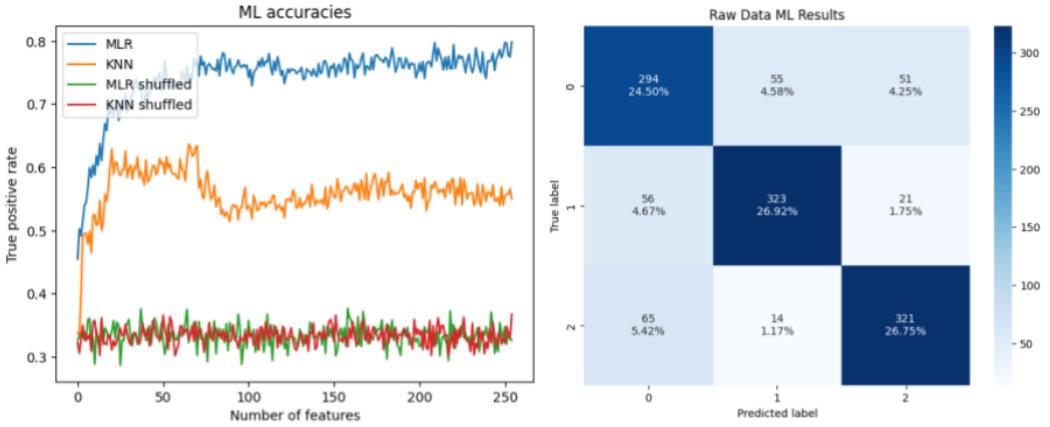


Figure 4.8: *Left*: Accuracies of a 10-fold repetition fitting of the 1NN and MLR models over the n features using the raw data. *Right*: Sum of the confusion matrices of the 10-fold repetition fitting of the MLR model.

Firstly, we revised how the raw data performed on raw data with just epoching performed on it. The MLR model performed substantially better with an average accuracy of 74.18% compared to the average accuracy of 51.21% given by the 1NN model. It was also observed that the shuffled labels also gave around the 33.33% expected accuracy of a random guess model. With the information presented by **Figure 4.8** we can see that the model is balanced when predicting the different classes. It is also noticeable that around 80 features the accuracy plateaus in the MLR model. Nonetheless, the 1NN model drops in accuracy around the 80 features, probably due to fitting noisy features that pull the model's clusters apart.

While fitting the data, and with a tolerance of 30 iterations without improvement of $\epsilon = 10^{-}3$, the optimized MLR model was found at 87 features with an accuracy of 78.17%.

Band-passing

Now, the pipeline has only band-passed the data at 1Hz to 500Hz. The MLR model performed slightly worse compared to the raw data with an average accuracy of 70.96% compared to the average accuracy of 74.18% of the MLR trained on raw data. Similarly, the 1NN model in this step performed slightly worse with a 49.42% accuracy compared to the previous 51.21% accuracy. This may be due to bandpassing removing some supportive information around the brain wave data but not doing anything to improve the cleanliness of the frequency bands where brain waves are typically found.

It was also observed that the shuffled labels also gave around the 33.33% expected accuracy of a random guess model. With the information presented by **Figure 4.9** we can see that the model is balanced when predicting the different classes. Similar to the raw data, the 1NN model drops in accuracy when fit to more than 80 features.

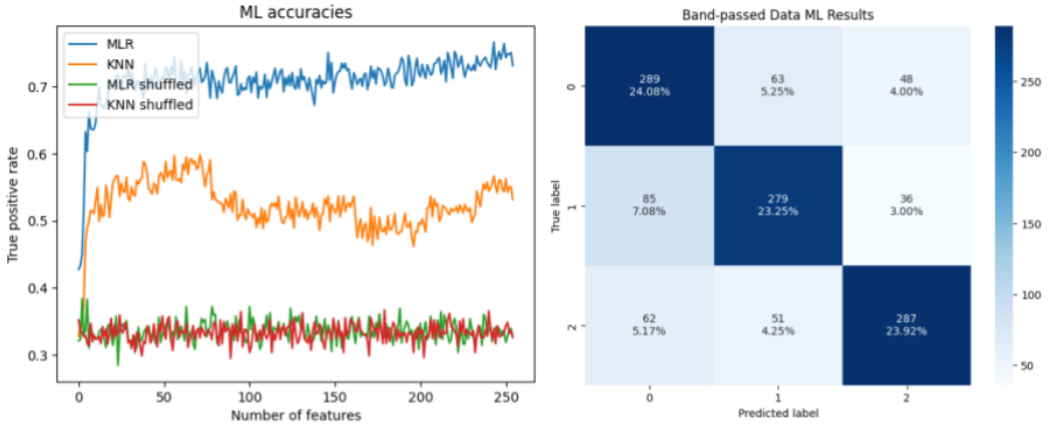


Figure 4.9: *Left*: Accuracies of a 10-fold repetition fitting of the 1NN and MLR models over the n features using the band-passed data. *Right*: Sum of the confusion matrices of the 10-fold repetition fitting of the MLR model.

While fitting the data, and with a tolerance of 30 iterations without an improvement of $\epsilon = 10^{-}3$, the optimized MLR model was found at 58 features with an accuracy of 74.5%.

ZapLine Filter

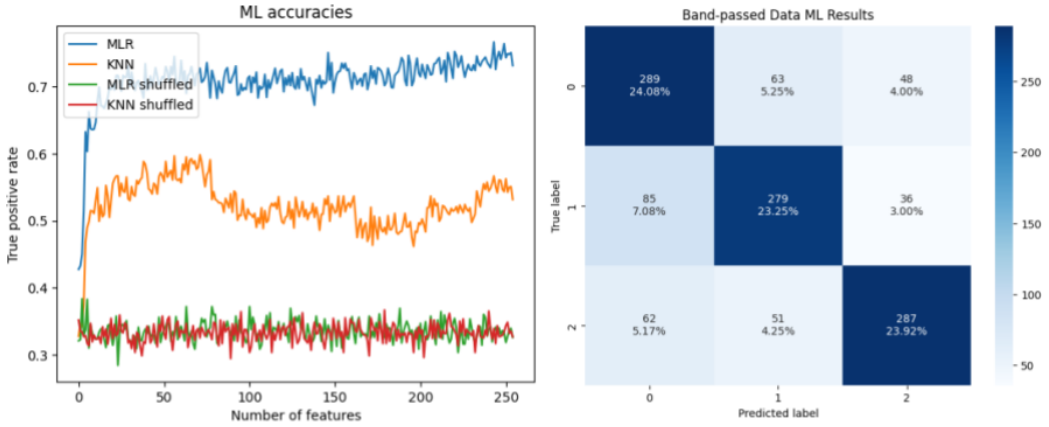


Figure 4.10: *Left*: Accuracies of a 10-fold repetition fitting of the 1NN and MLR models over the n features using the ZapLine filtered and band-passed data. *Right*: Sum of the confusion matrices of the 10-fold repetition fitting of the MLR model.

Subsequently, the pipeline applies the ZapLine filter at 60Hz to the previously band-passed data. Both models performed substantially better than the previous step, MLR had an average accuracy of 83.01% and 1NN had an average accuracy of 53.17%. With

this we can deduce that the ZapLine filter removes power supply noise effectively without removing important data needed for classification.

As per the shuffled labels, it was observed that accuracy rounded the expected 33.33% theoretical chance-level accuracy. With the information presented by **Figure 4.10** we can see that the model is balanced when predicting the different classes with no biases towards a single class.

While fitting the data, and with a tolerance of 30 iterations without an improvement of $\epsilon = 10^{-}3$, the optimized MLR model was found at 222 features with an accuracy of 89.25%. This also matches the best accuracy found in the whole iterations of number of features. This increment in number of significant features may be due to some channels that were heavily affected by power supply noise now being usable for classification.

Bad Channel Rejection

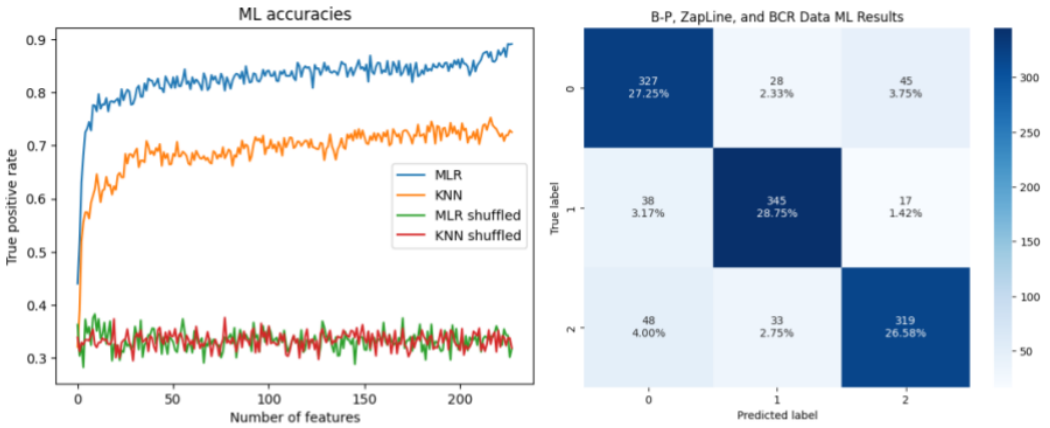


Figure 4.11: *Left*: Accuracies of a 10-fold repetition fitting of the 1NN and MLR models over the n features using the ZapLine filtered and band-passed data with bad channel rejection. *Right*: Sum of the confusion matrices of the 10-fold repetition fitting of the MLR model.

Next, the pipeline rejects the bad channels found in the ZapLine filtered and band-passed data. Both models performed statistically identical compared to the previous step, MLR had an average accuracy of 82.26% and 1NN had an average accuracy of 53.1%. With this we can see that fitting the bad channels from the models don't really affect the classification accuracy. Nevertheless, the time needed to train the model is reduced due to there being less features to fit.

As per the shuffled labels, the same 33.33% accuracy was seen. With the information presented by **Figure 4.11** we can see that the model is balanced when predicting the different classes with no biases towards a single class.

While fitting the data, and with a tolerance of 30 iterations without an improvement of $\epsilon = 10^{-}3$, the optimized MLR model was found at 227 features with an accuracy of

89.08%. This also matches the best accuracy found across all iterations of number of features.

ICA and ClusterMARA

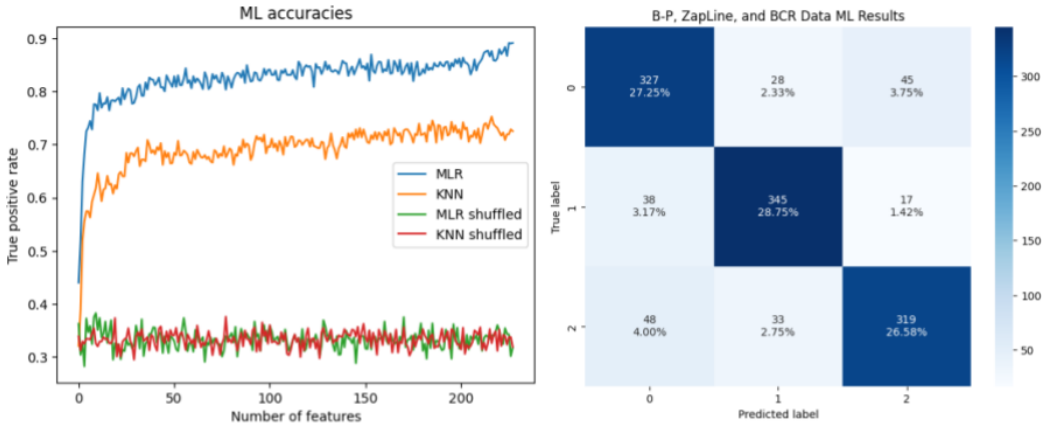


Figure 4.12: *Left*: Accuracies of a 10-fold repetition fitting of the 1NN and MLR models over the n features using the ZapLine filtered and band-passed data with bad channel rejection. *Right*: Sum of the confusion matrices of the 10-fold repetition fitting of the MLR model.

Finally, the pipeline removes the bad components using ClusterMARA from the previously processed data. Both models performed the best after cleaning all the data, MLR had an average accuracy of 92.7% and 1NN had an average accuracy of 76.1%. With this we can see that removing the bad components from the models enhances the overall classification power.

As per the shuffled labels, the same 33.33% accuracy was seen. With the information presented by **Figure 4.12** we can see that the model is balanced when predicting the different classes with no biases towards a single class. We also notice that both model have a constant upward trend in accuracies.

While fitting the data, and with a tolerance of 30 iterations without a improvement of $\epsilon = 10^{-}3$, the optimized MLR model was found at 212 features with an accuracy of 97.25%. This also matches the best accuracy found across all iterations of number of features.

4.3.2 Overall Performance of the Models

In **Figure 4.13** we can see that the overall performance of the Multinomial Logistic Regression performs substantially better than the 1NN. It was found the steps that impacted accuracy the most were the ZapLine and the ICA/ClusterMARA steps. This means that they are better at removing noise and artifacts and keeping important data for classification.

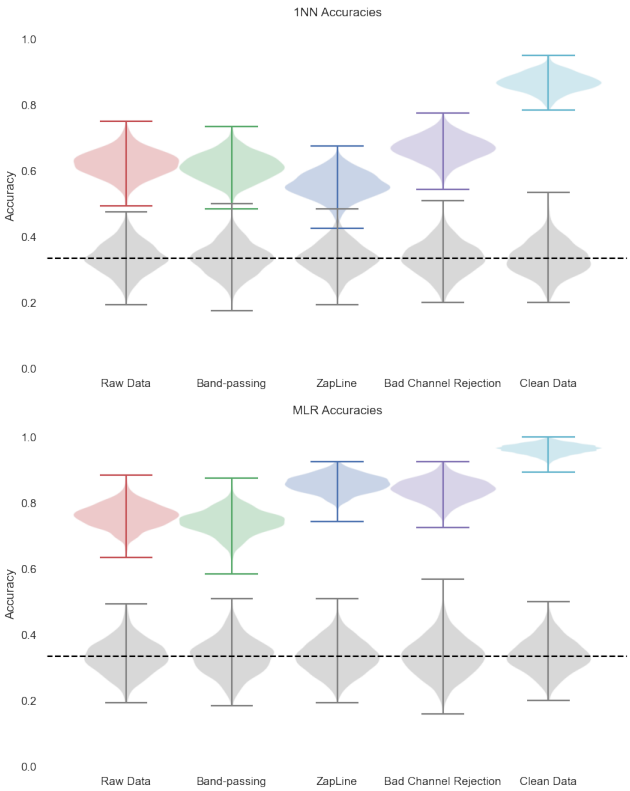


Figure 4.13: *Top*: Distribution of accuracies for the optimized models at each step with the 1NN models. *Bottom:* Distribution of accuracies for the optimized models at each step with the MLR models.

The best mean accuracy for the MLR optimised model was 97.25% at 212 features. For 1NN the best mean accuracy for the 1NN optimised model was 82.31

Chapter 5

Discussion

The goal of this project was to create an automated EEG preprocessing pipeline that was able to fit to multiple research use cases; another goal was to find a way to assess the cleanliness of EEG/LFP signals. We achieved these goals by creating NeuroClean, a multipurpose neural data preprocessing pipeline that uses 6 distinct steps to remove artifacts and noise from the EEG and LFP data. We designed and implemented a workflow that enables the pipeline to use standard EEG data and specific case by case classification task data to fit itself to the needed study and research case.

NeuroClean uses the 6 previously mentioned steps: Band-passing, which removes unwanted frequencies outside 1Hz and 500Hz; ZapLine filtering, a simple yet powerful way to remove power supply noise; Bad Channel Rejection, a method based on standard deviation was used to remove unusable channels; ICA and ClusterMARA, a modification of the Multiple Artifact Rejection Algorithm (MARA) was used to remove noisy artifacts from the signal; and finally epoching, given events during the experiments the data was segmented into classes.

The pipeline achieved an accuracy of 97.25% accuracy in a 3-class classification task after the data was cleaned. Comparing the result to the raw data accuracy 78.16%, we concluded that the pipeline is efficient at removing noise and artifacts, and at keeping key information needed for the given classification task. We believe that NeuroClean provides a useful workflow to achieve clean LFP and EEG data, nonetheless some improvements can be made.

5.1 NeuroClean Compared to Other Pipelines

Earlier in this project we reviewed the background on EEG cleaning pipelines. Other pipelines such as HAPPE and ADJUST were reviewed. Compared to NeuroClean, these pipelines are supervised and fit their respective cases only, i.e infant data. Additionally, the methods for quantifying the effect on the signal relied on statistical data about the signal, but not its effect on a classification task, unlike NeuroClean, that relies on user-defined classification accuracies to assess data cleanliness.

NeuroClean also uses this accuracy to fit itself to any use case with a promising accu-

racy bound. Other pipelines have been tailored specifically to each case, meaning that they may have slightly better performance. Even so, some data removal is acceptable because of the abundance of data.

5.2 Limitations

While we tried our hardest to properly create the pipeline, the current iteration of NeuroClean exhibits some limitations, which we aim at overcoming in future versions of the code. Specifically, the self fitting to different use cases doesn't work with all kinds of neural data, i.e fMRI, calcium imaging, spiking, etc. NeuroClean may remove real brain signal and data due to the definition of signal cleanliness depending on classification tasks, this may result in fitted NeuroClean models performing better in the given classification task than any others. That is to say, the pipeline may remove data not important to the given task.

On another topic, the processing of the EEG data is very time and memory consuming. During the creation of the pipeline, 32GB of RAM was needed to process 4GB of LFP data, as well as the process taking 2 and a half hours.

Lastly, the modified MARA (clusterMara) involved clustering and rejection of outliers. This means that if there are several artifactual components clustered together, they wouldn't get rejected. This is a problem we aim to solve in the future.

5.3 Future Directions

There are several improvements that we could implement in future works:

- Firstly, an enhanced cluster rejection algorithm could be implemented in the ClusterMARA section of the pipeline. This could be a permutation or iterative version of the rejection of clusters. A proximity or quantity of neighbors version could also be implemented, taking into account that most of the components should be brain data when reaching this point.

- A grid search of the parameters of the steps could be employed to fit the pipeline better to other cases. This would mean changing the tolerances of the bad channel rejections, changing the distances and number of components in the ClusterMARA, etc..

- On the same topic, the ClusterMARA algorithm uses just 5 features, some other features seen in the Winkler I. 2011 [6] paper could be employed to fine tune the NeuroClean pipeline.

- Lastly, other features for the classification could be used apart from the mean spectral amplitude, such as covariance of signals, autoregressive coefficients of the signals, among others.

5.4 Conclusions

In conclusion, the original goal of creating a multipurpose EEG preprocessing pipeline was achieved by the flexible steps that were chosen, with a mix of research and testing. We developed a promising workflow and pipeline that could achieve over a 97% accuracy, compared to the 74% accuracy using raw data. NeuroClean competes closely with other current solutions for the EEG and LFP data cleaning problem. We wish to continue working on this pipeline and review the previous mentioned limitations.

Chapter 6

Project Development

We organized the project in 2 week plans and meetings, but to not waste time in between the regular weekly meetings we also met and advanced the project in other fronts and sides to implement as many steps as possible. In **Figure 6.1** the evolution of the pipeline is shown throughout the 12 weeks of the project's development,

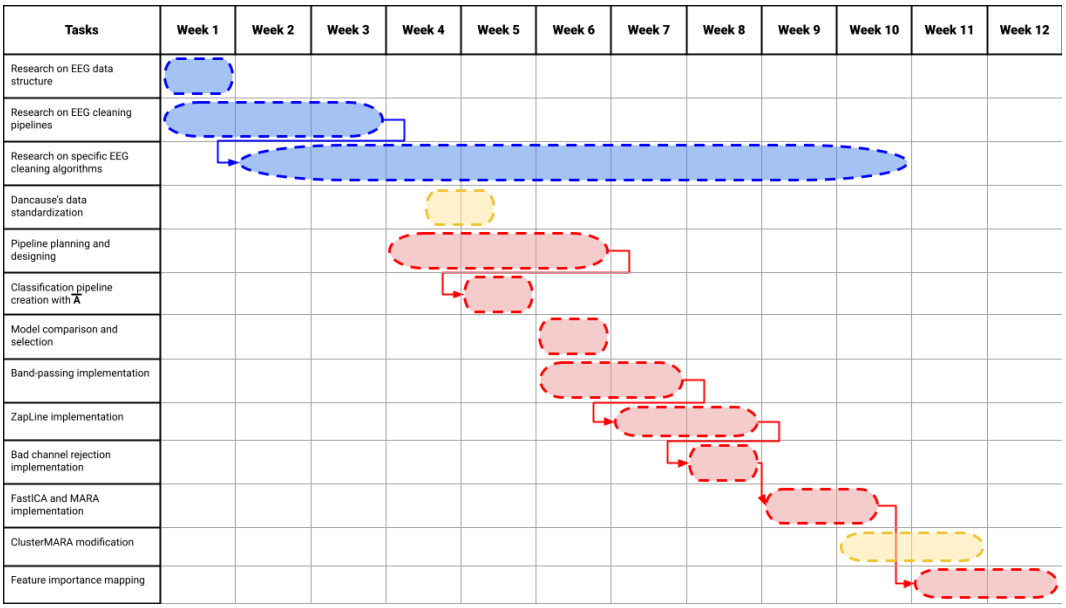


Figure 6.1: Distribution of work as shown with a Gantt Diagram. In red the critical path of the project, in blue supportive tasks, and yellow problems that arose while implementing the pipeline

Bibliography

[1] Jiang, X., G. B. Bian, and Z. Tian. *Removal of Artifacts from EEG Signals: A Review*, (2019), 19.5

[2] Laurel J. Gabard-Durnam, Adriana S. Mendez Leal, Carol L. Wilkinson and April R. Levin, *The Harvard Automated Processing Pipeline for Electroencephalography (HAPPE): Standardized Processing Software for Developmental and High-Artifact Data*, In: Front. Neurosci., **vol 12**, (2018).

[3] Ana Fló, Giulia G., Lucas B., Ghislaine D.-L., *Automated Pipeline for Infants Continuous EEG (APICE): A flexible pipeline for developmental cognitive studies*, In: Developmental Cognitive Neuroscience., **vol 54**, (2022).

[4] Chun-Hsiang Chuang, Li-Wei Ko, Yuan-Pin Lin, Tzyy-Ping Jung, Chin-Teng Lin, *Independent Component Ensemble of EEG for Brain-Computer Interface*, In: IEEE Transactions on Neural Systems and Rehabilitation Engineering. **vol 22**, (2013) 230-238.

[5] Mognon A., Jovicich J., Bruzzone, L., and Buiatti, M, *ADJUST: an automatic EEG artifact detector based on the joint use of spatial and temporal features*, In: Psychophysiology. **vol 48**, (2011) 229-240.

[6] Winkler I., Haufe, S., and Tangermann, M., *Automatic classification of artifactual ICA-components for artifact removal in EEG signals.*, In: Behav. Brain Funct.. **vol 7**, (2011)

[7] Nolan, H., Whelan, R., and Reilly, R. B. *FASTER: fully automated statistical thresholding for EEG artifact rejection.*, In: J. Neurosci. Methods. **192**, (2010), 152-162

[8] Bigdely-Shamlo, N., Mullen T., Kothe C., Su, K.-M., and Robbins K. A. *The PREP pipeline: standardized preprocessing for large-scale EEG analysis.*, In: Front. Neuroinform. **9:16**, (2015)

[9] F. Sandhaeger, Constantin v. N., Earl K. M., and Markus S., *Monkey EEG links neuronal color and motion information across species and scales.*, In: eLife. **8**, (2019)

[10] Tomoya N., H. Trong D., *Non-invasive electroencephalographical (EEG) recording system in awake monkeys.*, In: Heliyon. **6(5)**, (2020)

[11] Angel M., Raquel D., and Natividad D., *An Unsupervised Method for Artefact Removal in EEG Signals.*, In: Sensors (Basel). **19(10)**, (2019)

[12] Shaun D. Fickling, Careesa C. Liu, Ryan C. N. D'Arcy, Sujoy Ghosh Hajra, Xiaowei Song, *Good data? The EEG Quality Index for Automated Assessment of Signal Quality.*, In: 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), (2019)

[13] A. de Cheveigné, *ZapLine: A simple and effective method to remove power line artifacts.*, In: NeuroImage **vol. 207**, (2020)

[14] M. Komosar, P. Fiedler, and J. Haueisen, *Bad channel detection in EEG recordings.*, In: Current Directions in Biomedical Engineering, (2022), 257-260

[15] Butterworth S., *On the Theory of Filter Amplifiers.*, In: Experimental Wireless and the Wireless Engineer, (1930), 536-541

[16] S. S. Daud, and R. Sudirman, *Butterworth Bandpass and Stationary Wavelet Transform Filter Comparison for Electroencephalography Signal.*, In: 6th International Conference on Intelligent Systems, Modelling and Simulation, (2015)

[17] A. de Cheveigné, and L. C. Parra, *Joint decorrelation, a versatile tool for multichannel data analysis.*, In: Neuroimage, 98 (2014), 487-505

[18] **Meegkit** github main page, documentation and source code: `https://nbara.github.io/python-meegkit/index.html`

[19] A. Hyvarinen, and E. Oja, *Independent Component Analysis: Algorithms and Applications.*, In: Neural Networks, 13(4-5) (2000), 411-430

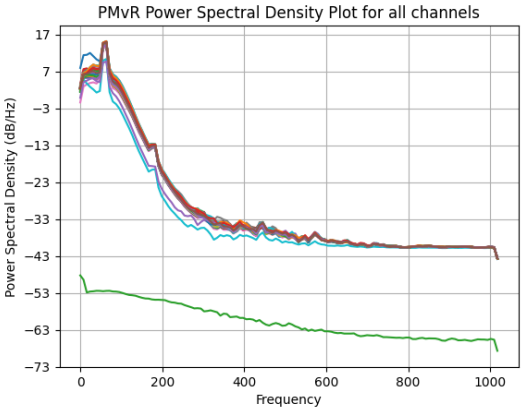
[20] D. Mika, and J. Józwik, *Single Channel Source Separation with ICA-Based Time-Frequency Decomposition.*, In: Sensors, 20(7) (2019)

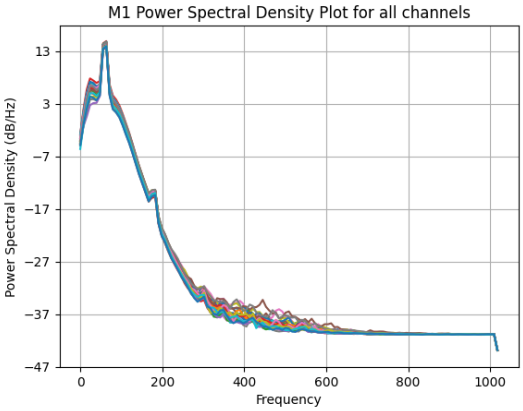
[21] Ester M., H. P. Kriegel, J. Sander, and X. Xu, *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise.*, In: Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, Portland, OR, AAAI Press, (1996), 226-331

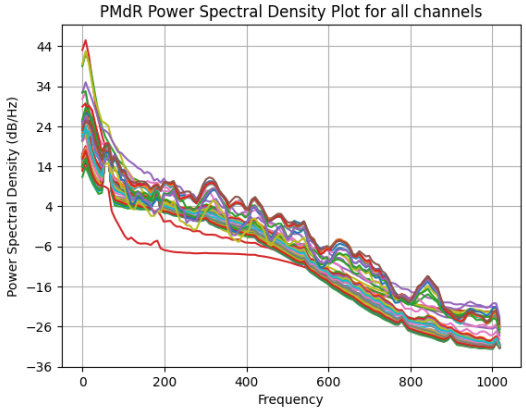
Brain area signal data

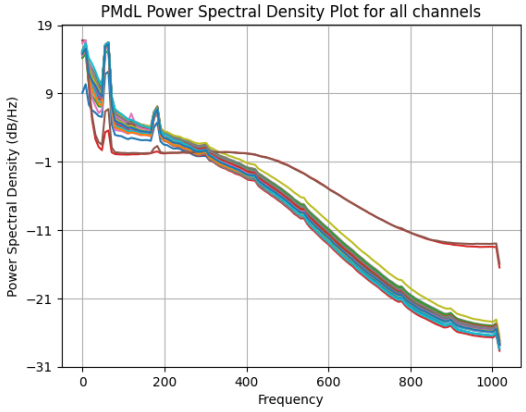
Power Spectral Density plots of all channels by brain area after each step.

Raw data

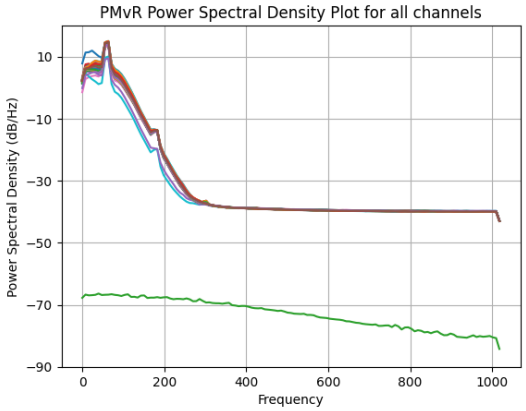


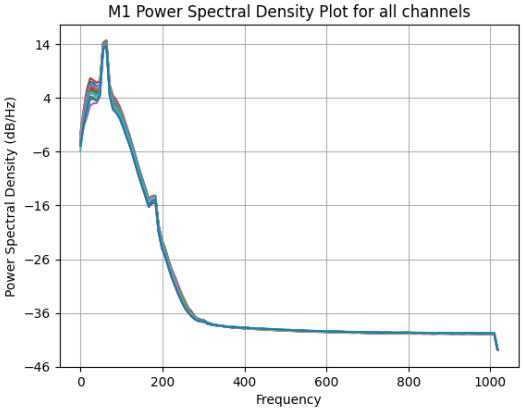


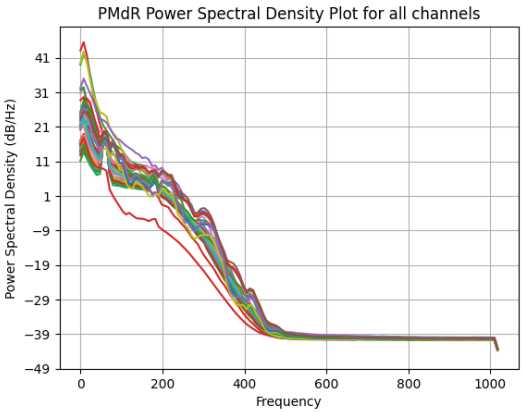


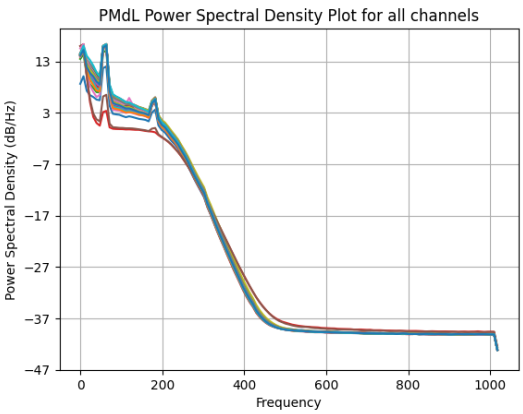


After band-pass

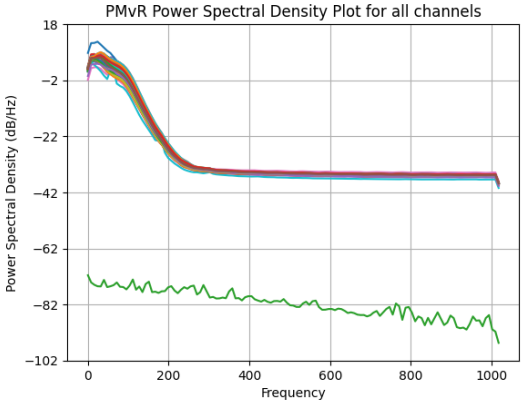


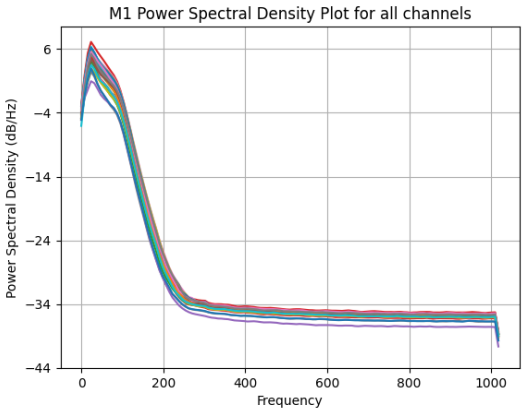

M1 Power Spectral Density Plot for all channels
Power Spectral Density (dB/Hz)
14
4
−6
−16
−26
−36
−46
0
200
400
600
800
1000
Frequency

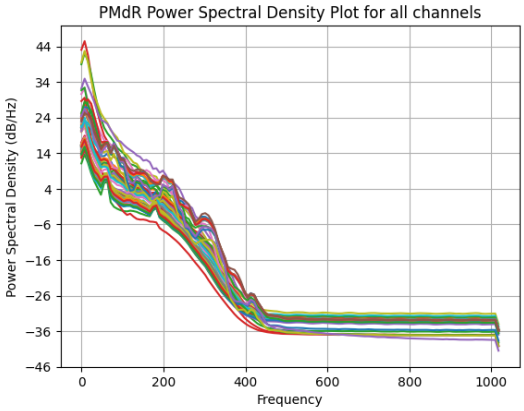

PMdR Power Spectral Density Plot for all channels
Power Spectral Density (dB/Hz)
41
31
21
11
1
−9
−19
−29
−39
−49
0
200
400
600
800
1000
Frequency

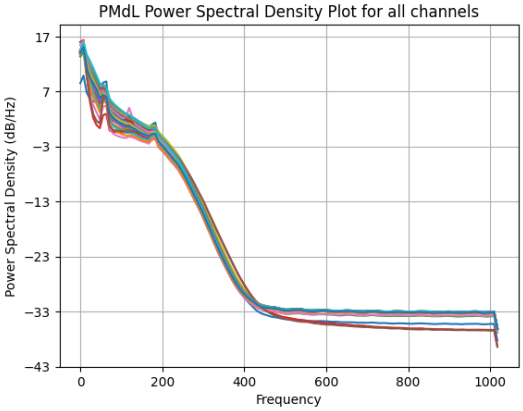

PMdL Power Spectral Density Plot for all channels
Power Spectral Density (dB/Hz)
13
3
−7
−17
−27
−37
−47
0
200
400
600
800
1000
Frequency

After band-pass and ZapLine

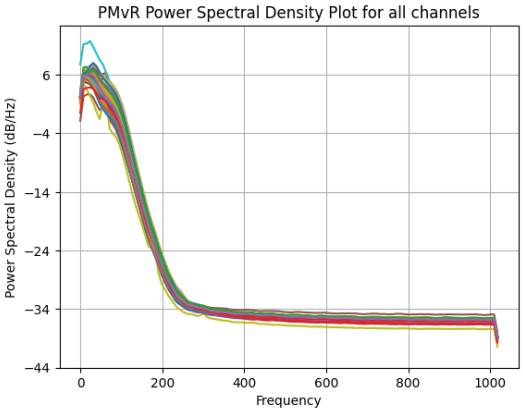


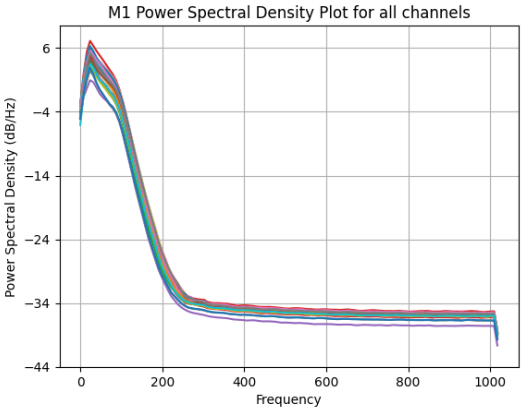


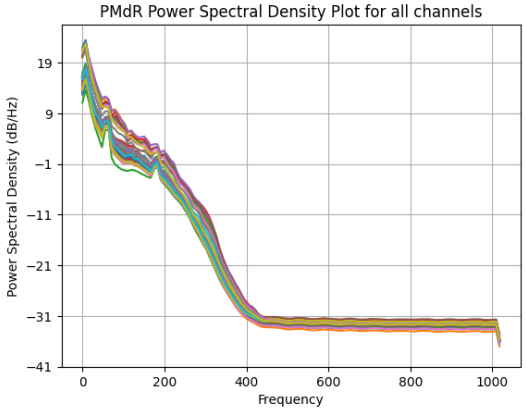


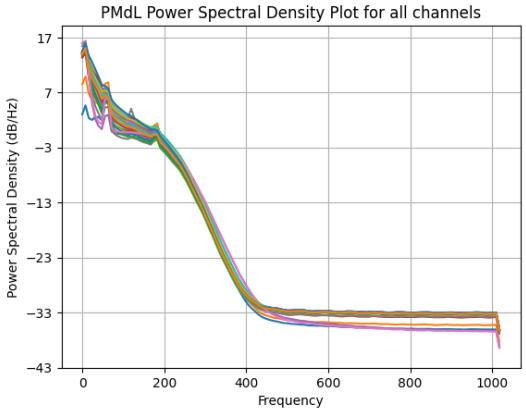


After band-pass, ZapLine and Bad Channel Rejection

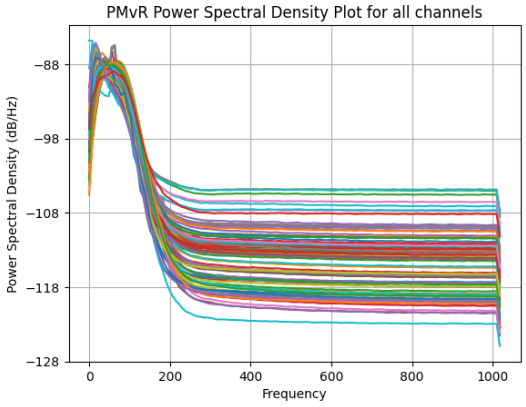


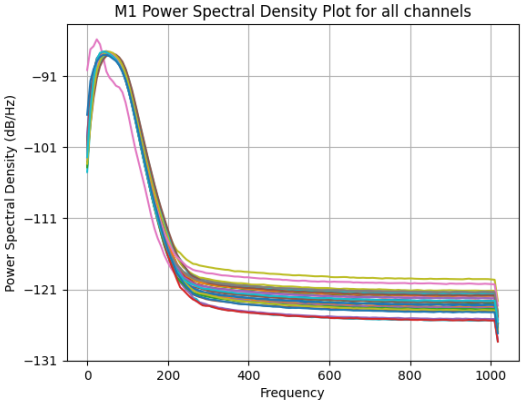


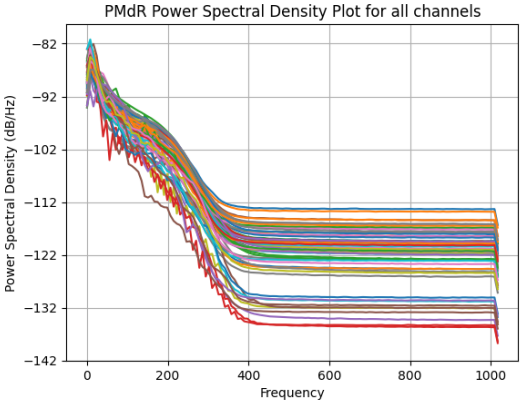


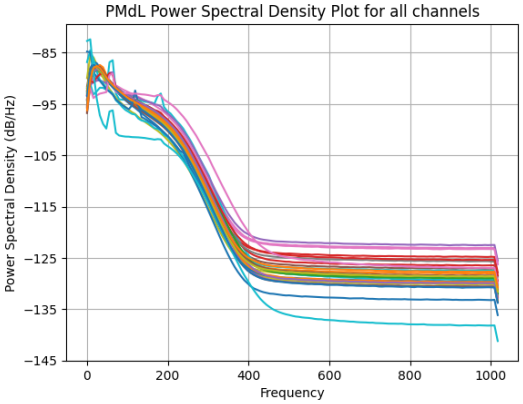


After the full pipeline




M1 Power Spectral Density Plot for all channels
Power Spectral Density (dB/Hz)
−91
−101
−111
−121
−131
0
200
400
600
800
1000
Frequency


PMdR Power Spectral Density Plot for all channels
Power Spectral Density (dB/Hz)
−82
−92
−102
−112
−122
−132
−142
0
200
400
600
800
1000
Frequency


PMdL Power Spectral Density Plot for all channels
Power Spectral Density (dB/Hz)
−85
−95
−105
−115
−125
−135
−145
0
200
400
600
800
1000
Frequency