Research Project - Creating Pipeline for Preprocessing of EEG Data

Project Code: V24KIREPRO1PE818

Laura Kazlauskaite
IT University of Copenhagen
lkaz@itu.dk

Amalie Katrine Kaa Mortensen
IT University of Copenhagen
akmo@itu.dk

Supervisor: Paolo Burelli
BrAIn Lab, IT University of Copenhagen
pabu@itu.dk

December 2024

1 Abstract

This report summarizes the development and evaluation of an EEG data preprocessing pipeline designed to support emotion recognition research[1]. The pipeline applies EEG data cleaning and preparation techniques, such as data visualization, bad channel detection, and independent component analysis (ICA) and outputs a report and a cleaned EEG file in the fif format. The pipeline utilizes algorithms for detecting bad channels based on flat channel values, high-frequency noise, signal deviation, correlation, signal-to-noise ratio, and the RANSAC method. Additionally, it offers the option to interpolate bad channels and perform ICA to isolate independent components, allowing for artifact detection. The tool is intended to be accessible to newcomers in EEG research, particularly those with limited prior experience in the field, and is customizable through a configuration file. Despite its current limitations, including the lack of automatic artifact rejection and filtering options, the pipeline has been successfully tested on EEG data from 49 participants. The results demonstrate the pipeline's capabilities in applying the cleaning techniques with the user's chosen parameters and generating a report and a EEG file. Further development for the pipeline is needed to improve its functionality, such as providing more descriptive plots, incorporating more filtering options, better artifact detection, and support for additional EEG file formats.

[1]The exact same code provided in the attached .zip file at submission can be downloaded from our public GitHub repository using the following link: `https://github.com/amaliekaa/eeg_preprocessing_pipeline.git`. After submission no further changes will be made in the repository until the 10th of January 2025.

2 Contents

Contents

3 Introduction

Electroencephalography (EEG) is a brain imaging technique developed a century ago that remains valuable and widely utilized for studying human brain activity[1]. By placing electrodes on the scalp, EEG captures electrical activity originating from the brain. A big advantage of EEG is its ability to capture hundreds of samples per second allowing for close monitoring of the brain activity over time. Being a non-invasive technique, EEG is particularly suitable for large-scale research projects aimed at understanding the activity of the human brain - a research field with much yet to be explored.

However, EEG signals are often contaminated with noise from sources such as muscle artifacts, eye blinks, and environmental interference. Noise can obscure meaningful brain activity making it harder to interpret. Noise reduction is a critical part of preprocessing EEG data, where the overall goal is to obtain the clearest possible signal that accurately represents brain activity. As machine learning and computational tools continue to improve, our ability to analyze data from large-scale studies has grown significantly. This makes it increasingly important to have standardized preprocessing tools. These tools facilitate collaboration across research labs and help ensure higher-quality data and more reliable findings[2, 3].

Challenges in preprosseing of EEG data are especially prominent in fields such as emotion detection, where little is known about how emotion shows in EEG recordings. With much yet to be explored the need of preserving even subtle patterns in EEG data at all possible frequencies, while reducing noise, is critical.

This project is conducted in cooperation with BrAIn Lab at the IT University of Copenhagen, which provided a dataset of EEG recordings from an experiment focused on emotion recognition. The project focuses on the development of a preprocessing pipeline useful for all EEG studies, but specifically tailored for emotion-related EEG data with the aim for enabling more accurate and efficient analysis using machine learning techniques.

3.1 Research Question

How can we design a preprocessing pipeline that cleans and prepares EEG data for analysis related to emotion recognition, while ensuring it is accessible and usable for researchers with minimal experience in EEG processing?

4 Theory

4.1 Electroencephalogram (EEG)

Electroencephalography (EEG) is a non-invasive technique used to record the brain's electrical activity. It is widely used in medical settings, for example to diagnose seizures and sleep disorders where it is important to have a high sampling frequency[1, 4] (number of samples per second). EEG is also useful

in brain research, where analyzing EEG data from groups of participants can provide insights into normal brain function and for example identify temporal patterns and active areas of the brain.

The brain contains billions of neurons, which communicate by releasing neurotransmitters that trigger electrical impulses. The neurons often fire in coordinated groups creating electrical fluctuations that can be recorded by placing electrodes on the scalp. While the brain is constantly active, the frequencies of these electrical signals vary depending on the type of activity, such as sleeping, moving, or concentrating. [4]

4.1.1 Brain wave frequencies

The brain wave frequencies are divided into ranges. While all ranges of frequencies are present at all times, each is more dominant in certain types of activities[4, 1, 5]:

- Waves of less than 4 Hz are called delta waves and are commonly associated with deep sleep stages.
- Waves of 4 - 7 Hz are called theta waves and are often observed during light sleep, relaxation, and are also linked to daydreaming and creativity.
- Waves of 8 - 12 Hz are called alpha waves, which are most prominent when one is awake and in a calm state.
- Waves of 13 - 30 Hz are called beta waves, which are associated with awake states such as focus, active thinking, performance, concentration, and higher cognitive tasks.
- Waves of 30 Hz and more are called gamma waves, which are associated with functions such as learning, memory and information processing. A high amount of beta and gamma waves is associated with states such as anxiety and stress. Some sources place the cut-off between beta and gamma waves at 35 - 40 Hz [1].

In order to benefit from the information on frequencies, temporal patterns and active areas in the brain we need to be able to systematically analyze the data collected in EEG recordings.

4.2 Bad channel detection

The signals in EEG recordings are captured as separate channels, with one channel corresponding to each electrode placed on the scalp. The electrodes are positioned according to a standardized placement system, enabling the software processing the data to identify the region of the brain each signal originates from. Electrodes can be placed individually or embedded in a cap to ensure consistent placement. [1]

However, electrodes can sometimes malfunction, and external electrical signals unrelated to brain activity can introduce noise into the recording. [3]

Some of this noise can be removed later by applying high-pass and low-pass filters to the channels, which set frequency ranges to exclude signals outside the desired spectrum. While this approach effectively reduces noise, it also carries the risk of removing valuable information from the data. [3]

Another way of removing noise from the data is by using tools like Independent Component Analysis to identify artifacts.

4.3 Independent Component Analysis (ICA)

Independent Component Analysis (ICA) is a technique used to separate different underlying sources within some given data[1, 6]. ICA identifies components that are statistically independent of one another, which increases the likelihood that each component represents a distinct real-world source. In ICA it is assumed that this is the case[1, 6]. In the context of EEG, these components could include artifacts (such as eye blinks, muscle movements, or drift from slight electrode movement), as well as the actual brain activity captured in the recording[1].

ICA assumes that all channels in the recorded data represent a linear mixture of the original sources, and that these sources are statistically independent of each other. For optimal performance, ICA requires that the number of channels is equal to or larger than the number of sources. [1, 6]

To illustrate the concept of ICA, consider the case of microphones recording sounds in a room. If multiple microphones are picking up audio from multiple people talking, along with background noise, ICA can be applied to separate the mixed signals into distinct components, ideally isolating each voice (source) and the noise, based on their independent patterns. This principle mirrors the way ICA separates brain activity from noise (artifacts) based on their independent characteristics. [6]

Identifying the underlying sources allows for effective artifact removal before reconstructing the channels for further analysis[1]. However, ICA itself does not automatically tell us which components correspond to artifacts and which represent brain activity. This requires visual inspection or automated classification methods to distinguish between the two.

5 Related Work

5.1 MNE-Python

MNE-Python [2] is an open source package software that aims to provide tools for processing EEG and MEG data, allowing the user to create pipelines in Python scripts. It makes use of well established scientific libraries such as NumPy, SciPy, matplotlib and Mayavi. Using the MNE-Python package you create `raw` objects, which are instances of unprocessed EEG data derived directly from the provided EEG recordings. These files include all data collected during an EEG recording and serve as the foundation for subsequent processing. Any future mention of MNE that is not explicitly referenced is referring to

MNE as found in this MNE package[2], and whenever we refer to a `raw`, it is the MNE `raw` object.

5.2 PyPrep

The PyPrep pipeline was developed to standardize the early preprocessing of EEG data [3]. The pipeline is designed to work with MNE. Given raw EEG data in MNE's `raw` object format, the pipeline does preprocessing such as removing line noise (electrical interference from power lines or equipment), adjusting the reference channel to improve signal quality, and detecting and interpolating bad channels[3]. This pipeline is suitable for cleaning individual EEG recordings but does not provide functionality for handling and processing large datasets across multiple subjects.

Along with the pipeline for cleaning single EEG files PyPrep has also released separate functions for detecting bad channels and re-referencing[8, 9].

5.3 Brain Imaging Data Structure (BIDS)

The Brain Imaging Data Structure (BIDS) was created to standardize how data collected using modern brain imaging techniques is organized and described, facilitating better collaboration across studies and research labs [10]. While initially developed for Magnetic Resonance Imaging (MRI), BIDS has since been expanded to include a variety of other brain imaging techniques, including EEG[11].

BIDS specifications define the naming conventions and organization of folders containing participant data. The structure is designed to enable pipelines to automatically extract the desired files and information from a BIDS-compliant dataset without requiring manual intervention.[10]

6 Methods

We have decided to use the MNE and PyPrep[3] libraries to setup our preprocessing pipeline. MNE offers a comprehensive toolkit for processing EEG data, and PyPrep, while building on MNE, adds further functionality and standardization specifically related to preprocessing. Both are aiming to set standards for research on EEG data which make them good choices for our project.[2, 3]

Additionally, our pipeline is implemented to work on datasets following the BIDS format. By optimizing for this format, we aim to ensure that the pipeline is living up to the growing need for standardization and can be utilized by a broad range of researchers.

The user interface to the pipeline is the `config.ini`, our configuration file. The user needs to insert the chosen parameters and run the `Main.py` file for the pipeline to execute. The pipeline outputs a report for each participant, which can then be used to inspect the data. In addition, it outputs a log file with

terminal output from all functions run in the pipeline to ease troubleshooting and a cleaned EEG file in the .fif format.

To run the pipeline some parameters has to be set in the configuration file. The general parameters are the following:

- Dataset path: Mandatory. Path to where the data is located. The data must be organized according to the BIDS format.
- File extension: Optional. Defaults to .edf. The extension of the EEG data files. There are several file formats for EEG data. Our pipeline at this stage only supports the .edf format. If a different format is given, the user needs to implement the conversion to MNE's raw format themselves and a terminal output is made telling the user to do so.
- EEG placement scheme: Optional. Defaults to "standard_1020". The scheme defines where on the scalp the electrodes are positioned.
- Reference channel: Optional. If the reference channel has not been set in the provided EEG data, then it must be specified here. If not specified, the pipeline infers the data provided had already applied a reference channel.
- Line frequency: Optional. Defaults to 50. The power line frequency of where the data was collected. Normally 50 or 60 Hz depending on the region.
- Run ICA: Optional. Defaults to False. Choose whether ICA should be run. ICA takes a while to run, thus depending on the user's needs, they can choose to include or exclude it.
- Interpolate bad channels: Mandatory. Select if the bad channels should be interpolated or not.
- Save EEG as fif: Mandatory. Select if the cleaned EEG file should be saved as a .fif file.

The other parameters are specific to bad channel detection, and thus will be explained in the corresponding section.

When the pipeline is run from `Main.py`, the data in the given dataset path is read and objects for pre-processing are created. The `read_directories.py` file contains the functions used for reading the data and creating objects.

Objects of type `Participant`, `Session` and `Run` are created, where their respective data is stored.

The `Participant` object stores data about the participant, such as their id, path to participants folder and a list of `Session` objects the participant has. If only one session was recorded, the one session is stored.

The `Session` object contains data about the session. Some of the information stored in this object is session id, the reference to `Participant` object and the list of `Run` objects belonging to that particular session.

The `Run` object, similarly, contains attributes, such as a reference to a `Session`, the filepath to the EEG file, run id and information about bad channels found.

We have chosen to define these custom classes to store various attributes, as this approach proved to be the most optimal for our needs.

After the objects are created, the pipeline is executed for all participants according to the selected parameters. For each participant, the pipeline generates a dedicated folder and produces a report for every recorded session. If a participant has only one session, a single report is created. Within each report, the paths to the runs are printed, allowing users to identify the run id associated with each run.

Then, `pipeline_manager.py` calls various preprocessing functions defined in `cleanup_functions.py`. The following sections describe the preprocessing steps performed, in the sequence they appear in the generated session reports.

The following steps describe the methods of the pipeline in detail. It should be noted that we have decided not to implement lowpass and highpass filters to the pipeline. While these filters are commonly used when cleaning EEG data, we have chosen not to filter out any particular frequencies. Since our project is meant to prepare the EEG data for emotion detection and so little is known about where in the EEG recordings emotion can be seen, we have decided to keep as much original data as possible.

Looking into recent research aiming to explore emotional patterns in EEG data we find no studies that allow frequencies below 1 Hz, but the only explanation given is to reduce noise. Even though including the lower frequencies does impose a high risk of allowing noise into the data, we are curious to see whether the further data analysis would be impacted positively or negatively by this decision when identifying emotional patterns.

6.1 Step 1 - Raw file creation

The pipeline starts with the generation of `raw` objects. Each `raw` object corresponds to a specific run and is stored within the `Run` object. Because `raw` objects are frequently utilized in later processing stages, they are created and saved for every run. Right after creation of the `raw` object the selected electrode placement scheme is set.

Within step 1, a sensors graph is generated and included in the session report. This graph provides a visualization of sensor placements on the scalp, based on the selected scheme. While it does not precisely depict the actual electrode positions on each participant's scalp, it serves as a helpful tool for users to understand the general layout and positioning of the electrodes.

The raw information for each run is then printed using MNE's raw.info(). This information is taken directly from the data before any processing is applied. The following attributes are printed in the report:

- Number of channels: the number of channels recorded during a run, this corresponds to the number of electrodes on the EEG equipment.

- Channel list: the list of the names of all the channels recorded.
- Highpass: all frequencies that are below the printed threshold are not part of the recording.
- Lowpass: all frequencies that are above the printed threshold are not part of the recording.
- Sampling frequency: sampling frequency specifies the average number of samples obtained in one second.

6.2 Step 2 - Generating raw channel plots

In step 2, we use MNE's functionality to create raw channel plots to visualize the collected data. These plots provide an overview of the unprocessed signals recorded from each channel over time, displaying voltage fluctuations (in microvolts, µV) captured by each electrode during the recording. These fluctuations reflect brain activity, noise, or physiological artifacts such as eye blinks or muscle movements. Each channel represents a specific scalp electrode, arranged in rows and labeled accordingly on the graph.

Artifacts like eye blinks, muscle activity, or external noise may sometimes be visible in the plot, but identifying them often requires further processing. The raw channel plot serves as a valuable tool for a quick assessment of signal quality, helping to detect flat, noisy, or irregular channels that could indicate issues such as poor electrode contact, hardware malfunctions or interference.

The plot included in the report provides a preview of the first five seconds of the recording, giving a quick overview of the channels and a general sense of how the signals appear. However, for more detailed visual inspection of the raw data, it would be beneficial to have an interactive graph or a longer preview of the channels. We have noted this as a potential improvement for future versions of the pipeline.

6.3 Step 3 - Power Spectral Density plots

Step 3 uses MNE's functionality for creating Power Spectral Density (PSD) plots for each run and includes them in the report. A PSD plot illustrates how power is distributed across different frequencies in the recorded data. Essentially, it shows the intensity of the signal at each frequency across all channels.

Often, the PSD plot will display noticeable noise at the frequency corresponding to the line frequency (typically 50 or 60 Hz, depending on the region). The user specifies this frequency in the configuration file. To remove this noise, we apply a notch filter by MNE that eliminates data at that specific frequency. After filtering, another PSD plot is generated and added to the report showing the cleaned data.

These plots are useful for visually identifying the most active frequencies and can also help pinpoint noise sources.

6.4 Step 4 - Bad channel detection

Step 4 focuses on identifying bad channels in the EEG data. A bad channel is one that does not provide reliable or meaningful data due to issues such as poor signal quality, bad connections, artifacts, or malfunctioning equipment. Identifying these channels can be crucial to ensuring that the data used for analysis is both accurate and of high quality. Bad channels can exhibit characteristics like excessive noise, flatlines, large spikes, or very low correlation with other channels. Since there are various reasons a channel may be considered bad, multiple functions are used to identify these channels. In this step, we utilize six different functions defined in the PyPrep[3] library to identify bad channels, and we will explain each of them in detail.[3, 9]

In the configuration file, users can choose which algorithms for identifying bad channels to include and can also define parameters for each algorithm, if desired. Users can specify True or False for each algorithm under the "Select algorithms run" section in the configuration file. If no selection is made, all algorithms default to False. It is important to note that, regardless of the user's configuration, all algorithms will still execute, and the report will show a list of the channels identified as bad by each algorithm. However, only the bad channels flagged by the algorithms marked as True will be included in the final list of bad channels for each `Run`. Subsequent preprocessing steps will consider only these channels as bad.

Below we aim to explain in simple language what each algorithm does. We have also included a more mathematical explanation of each algorithm as defined in the PyPrep library in order be able to explain how the parameters can be adjusted if the user wishes to do so. However, the focus of this paper and overall project is not to deep dive into the mathematical formulas and explain them in detail but rather to be able to understand and apply the algorithms.

6.4.1 Bads by nan flat

This algorithm identifies channels that either contain NaN values or have nearly flat signals. According to PyPrep library, "a channel is considered flat if its standard deviation or median absolute deviation (MAD) from the median is below the specified threshold for flatness (default: 1e-15 volts)." [9]

This PyPrep library method sets the threshold very low, which means that only channels that are nearly flat will be flagged as bad using this method. The threshold is set by default in the library without the option of changing it.

It must be noted that this method is always executed when any other method is run to ensure that nearly flat or nan channels do not interfere with other algorithms finding bad channels. [9]

6.4.2 Bads by hf noise

This algorithm detects channels that contain abnormally high amounts of high-frequency noise.[9]

The "noisiness" of a channel is measured by comparing the amplitude of its high-frequency components (above 50 Hz) to its overall amplitude. A channel is flagged as "bad-by-high-frequency-noise" if its noisiness is significantly higher than the median noisiness of all channels. This is determined using a robust Z-scoring method, with a specified Z-score threshold. Additionally, this method will only run bad channel detection if the signal's sampling rate is greater than 100 Hz. [9]

Parameters:

- hf_zscore_threshold: Optional, defaults to 5.0. Specifies the minimum z-score for which a channel should be considered bad-by-high-frequency-noise. [9] Increasing the threshold will make the algorithm less sensitive to high-frequency noise, potentially flagging fewer channels as bad. While decreasing the threshold will potentially flag more channels as bad.

6.4.3 Bads by deviation

This algorithm detects channels that have abnormally high or low overall amplitudes. [9]

A channel is labeled as "bad-by-deviation" if its signal strength (amplitude) is significantly different from the average signal strength of all channels. To measure this difference, the algorithm calculates how far each channel's amplitude is from the median amplitude, using a robust Z-scoring method that is resistant to outliers. If the difference exceeds a specified threshold, the channel is flagged as bad[9].

Parameters:

- deviation_threshold: Optional, defaults to 5.0. Specifies the minimum absolute z-score for which a channel is considered bad-by-deviation. [9] Increasing the threshold will make the algorithm less sensitive to deviation, potentially flagging fewer channels as bad. While decreasing the threshold will potentially flag more channels as bad.

6.4.4 Bads by correlation

This algorithm detects channels that do not correlate with any other channels and channels with periods of flat signal, based on given parameters.[9]

The algorithm first divides the data into small windows of time (default: 1 second) - correlation windows. For each correlation window, it calculates how strongly each channel's signal matches the signals of the other channels. The algorithm then picks the highest correlation for each channel in that time window.[9]

If a channel's highest correlation with any other channel is below the correlation threshold (default: 0.4), that time window is considered a bad correlation window for that channel. If there are too many bad correlation windows for

a channel (default: 1%), the channel is flagged as "bad-by-correlation." This means the channel is not consistently working well with the other channels.[9]

The algorithm also checks if a channel has flat signal for any periods. If the channel's signal is completely flat for a significant number of time windows (the same fraction threshold applies), it is flagged as "bad-by-dropout".[9]

Parameters:

- correlation_secs: Optional, defaults to 1.0. The length of each correlation window, measured in seconds.
- correlation_threshold: Optional, defaults to 0.4. The lowest correlation between a channel and all other channels for a channel to be considered "bad" within a given window.
- frac_bad_corr: Optional, defaults to 0.01 (1% of all windows). The minimum fraction of bad windows for a channel to be considered "bad-by-correlation" or "bad-by-dropout".[9]

6.4.5 Bads by SNR

This algorithm detects channels that have a low signal-to-noise ratio.

Channels are considered to have a low signal-to-noise ratio if they are bad by both high-frequency noise and bad by low correlation.

This algorithm does not take any additional parameters. It runs the previously described "bad-by-hf-noise" and "bad-by-correlation" algorithms based on their parameters and returns a list of channels that have been flagged by both of these algorithms.[9]

6.4.6 Bads by RANSAC

This method uses a technique called RANSAC to predict what the signal for each channel should look like, based on the signals and positions of other currently good channels.

The recording is divided into time windows (default: 5 seconds), and within each window, the method compares the actual signal of a channel with the signal predicted by RANSAC.

A channel is considered "bad" in a given window if the predicted signal does not match the actual signal well enough, based on a set threshold (default: 0.75). If too many of these "bad" windows occur for a channel (more than a given threshold, default: 0.4), that channel is marked as "bad-by-RANSAC."

Because RANSAC uses random sampling, the channels flagged as "bad" may vary slightly each time the method is run. Also, the bad channels might change depending on the electrode arrangement, since RANSAC's predictions depend on the positions of the electrodes.

It is important to note, that to ensure optimal performance, the channels bad by deviation and correlation must be flagged before running RANSAC. This

is implemented in the pipeline and correct performance is ensured regardless of which algorithms the user has set to use for bad channel detection. However, the list showing bad channels flagged by RANSAC will always contain the channels flagged by bads by deviation and bads by correlation.[9]

Parameters:

- n_samples: Optional, defaults to 50. Number of random samples to be used by RANSAC to predict the signal for each channel.
- sample_prop: Optional, defaults to 0.25 (25% of all channels). Proportion of channels to be used for signal predication.
- corr_thresh: Optional, defaults to 0.75 The minimum correlation between the predicted signal and the actual signal for a channel to be considered "good" in each RANSAC window.
- frac_bad: Optional, defaults to 0.4 (40% of all windows). The minimum fraction of bad windows for a channel to be considered bad-by-ransac.
- corr_window_secs: Optional, defaults to 5.0 (seconds). The length of each ransac window, measured in seconds.
- channel_wise: Optional, defaults to False. This option decides how RANSAC will predict the signals. If set to True, it will predict signals for chunks of channels over the entire signal length (channel-wise RANSAC). If set to False, it will predict signals for all channels at once but over smaller time windows (window-wise RANSAC). Channel-wise RANSAC generally needs more memory but can be faster if there's enough RAM.[9]

6.5 Step 5 - Bad channel interpolation

Once bad channels have been detected, we generally either exclude them from further analysis or interpolate them. While in some cases removing bad channels is sufficient, often, however, especially in cross-subject analysis, it is helpful to keep the same data dimensionality across all subjects. One option is to exclude the channels for all subjects where at least one subject has flagged that channel as bad. However, this is likely to result in a significant loss of good data. To avoid that, we can instead choose to interpolate bad channels. This means that the bad channels are reconstructed using the signals of the good channels. [12]

This is what step 5 of our pipeline does. It only runs if the user specifies "True" under the "interpolate_bad_channels" option in the configuration file. If "False" is selected, this step is skipped, and all channels are retained in their original state.

During this step, the built-in MNE functionality is used to interpolate channels marked as bad for each run. The interpolated signals overwrite the corresponding channels in the original `Raw` object for each run. MNE applies a

spherical spline interpolation method by default, which reconstructs bad channels using data from neighboring good channels.[12]

Interpolating bad channels means reconstructing them from their surrounding channels. Thus, for a good quality interpolation, we need to ensure that there exist good channels from which data can be interpolated. The interpolation of a channel can only be as good as the quality of the surrounding good channels, and will never be an accurate representation of the original data point. If many channels in an area are marked as bad this will decrease the quality of the interpolated channels.

6.6 Step 6 - ICA

Step 6 of the pipeline uses MNE's functionality to perform Independent Component Analysis (ICA). This technique decomposes the EEG channels into independent components, as described in Section 4.3. If the user has chosen to interpolate bad channels, ICA will process all channels, including the interpolated signals for previously identified "bad" channels. If interpolation is set to False, ICA will exclude the "bad" channels and analyze only the remaining "good" channels.

Step 6 generates two MNE plots for each run, displaying the mapped independent components and includes them in the report. The first plot shows a time-series representation of the individual components and the second shows a topographical representation of the same components. These plots can then be used for inspection and artifact detection.

6.7 Step 7 - Save Raw as fif

Step 7 in the pipeline provides the option of saving the processed `Raw` object as a .fif file in each participants folder using the MNE .save()[2] function. This step can be useful if the user needs the cleaned EEG file for further processing. If a file with the same name already exists in the directory, it will be overwritten.

7 Results

The EEG data preprocessing pipeline developed in this project provides a straightforward workflow to address common preprocessing tasks, particularly to users new to the EEG data analysis. The pipeline has been tested on a dataset with 49 different participants having EEG data. The pipeline successfully processed the dataset, creating a dedicated folder for each participant. Each folder contains session-specific reports, plots, and a cleaned EEG file, if the user chose to save it. The results below summarize the output of the pipeline. We will explain the results obtained during testing but will not include any of the graphs in the report to preserve the privacy of the study participants.

The pipeline successfully reads data from the specified BIDS-compliant dataset path and generates initial plots for visualization. The plots provide users with

an overview of their data and help identify any apparent issues. The initial plots show an overview of how the EEG sensors are placed on the scalp based on the chosen EEG placement scheme. The pipeline has been tested only on the "standard_1020" scheme however, and we cannot guarantee functionality if another scheme is chosen.

The next plot in the report is the raw plot showing an overview of the first five seconds of the recorded EEG data for each channel.

Next comes a plot illustrating the Power Spectral Density (PSD) for each run, allowing the user to see the signal intensity distribution across different frequencies. When testing on our data, the plot showed a spike at 50 Hz, which was expected as this is the line frequency in our region. Right after, another PSD plot is added showing that the notch filter has been applied filtering out the noise at 50 Hz. The user can specify the frequency of the line noise based on their region.

The next step in the pipeline performed bad channel detection using six algorithms from the PyPrep library. Channels containing flat signals, high-frequency noise, low correlation with other channels, or low signal-to-noise ratios were flagged as bad. Additionally, the RANSAC algorithm was used to cross-validate bad channel detection, identifying discrepancies between predicted and actual signals. In the tested data, different channels were flagged as bad across various runs. A common pattern we have noticed was that RANSAC consistently identified the most bad channels, suggesting it is relatively aggressive in finding bad channels. While the channels flagged by RANSAC may indeed be bad channels, we would suggest that the user carefully adjusts the parameters, should they choose to include this algorithm in their data cleaning process. The final list of bad channels was compiled based on user-selected algorithms and was included in the report under headline "Channels flagged as bad" .

When the option to interpolate bad channels was selected, the pipeline applied MNE's functionality for interpolation. While we have not deep dived into the quality of interpolated channels, we observed that the signal was successfully reconstructed for the identified bad channels.

If "run_ica" was selected as "True", ICA was applied to decompose the EEG data into independent components for artifact detection. Graphs of the independent components were generated and included in the report both as a time-series plot and as a spatial representation of the components. While this allows the user to inspect the independent components and identify artifacts, the functionality to remove or automatically detect artifacts was not included in this pipeline.

The pipeline successfully saved the preprocessed EEG data as .fif files for each run, providing users with cleaned data for further processing or analysis.

8 Discussion

8.1 User Interface

Despite the availability of standardized tools for EEG preprocessing, newcomers to the field such as ourselves might find it difficult to navigate the landscape of heavy literature and understand how to implement these tools effectively. This complexity can potentially hinder researchers who are primarily focused on analyses, such as applying machine learning to EEG data. When starting this project we chose to design it with fellow students in mind, making a tool for preprocessing of EEG data that would be easily accessible for students with some background in Computer Science but little to no knowledge about EEG. We hoped to build a tool that would allow students to jump almost directly to applying machine learning techniques to EEG data in order to gain deeper knowledge of the human brain rather than spend months learning how to effectively preprocess their data. We successfully made our pipeline easily accessible, and some important details for accessibility are:

- To run the pipeline all you need is a to add the path to a BIDS compliant dataset in a configuration file and make sure the correct placement scheme and file format are chosen.
- The configuration file has designated sections for each type of parameter and all parameters have clear and concise names along with necessary comments, allowing new users to set the parameters correctly based on the details of their dataset.
- The pipeline comes with a readme file that explains the most important steps for running the pipeline and also has a link to this report for further details.
- The user has the option to generate .fif files of the cleaned EEG data for further analysis. Also, a report and a log file are generated for each preprocessed run.

8.2 Drawbacks and Future Work

While tailoring the interface for fellow students, it would have been ideal to have designed a pipeline also useful for higher level researchers. However, there are some drawbacks of our current pipeline that make it less likely that it at its current state would be useful in larger research settings. Here are some of the improvements that would be needed in order to make the pipeline more useful:

- An option to view a larger section of the raw channel plots in order for them to be more useful in artifact detection.
- Implementation of the option to apply low- and high-pass filters to the data. This will be discussed in more detail below.

- An option to manually mark channels and components as bad after inspection of raw channel plot and ICA component plots, giving the option to reject these components before reconstruction of channels.
- Functionality for automatic artifact detection to reject artifacts such as eye blinks and muscle movements.
- An option to perform ICA before interpolation of bad channels, as there are cases with a large number of bad channels where interpolation prior to ICA might not be ideal.
- An option to set all parameters for all functions, allowing for more customization to meet the needs in each research case.
- Support for more EEG file formats.
- Proper testing on more datasets, including datasets with other placement schemes than the "standard_1020" and other EEG file formats than .edf.
- Support for identification of and the option to split recordings into epochs.
- Options to re-reference, for example the option to apply an average reference.
- Option to choose a path for the report and option to direct all derived data to derivatives folder, following BIDS methodology.
- To improve performance with large datasets, it would be ideal to include an option to resume the pipeline from where it left off after a crash, rather than starting over. Additionally, providing an option to process a specific sublist of participants would also be beneficial in larger settings.

These are clear areas of future work for this pipeline.

With the use case of emotion detection in mind we chose not to implement high- and lowpass filtering. This decision was based on the general lack of knowledge about what frequencies would include emotion associated data. To clarify whether especially highpass filtering is needed, the pipeline should ideally be tested with and without applying such filter using a machine learning model, so model performance could be assessed both with and without filtering.

This project aims not only to develop a preprocessing pipeline but also to document the decision-making process, creating a resource for researchers seeking clarity on how to approach EEG preprocessing for emotion recognition studies.

While implementing functionality provided by PyPrep, we only later discovered that PyPrep includes a pipeline for cleaning individual EEG files. Looking into this pipeline, it appears to be less customizable compared to the functionality we implemented in our own pipeline, supporting our decision to implement individual functions rather than relying on the full pipeline. However, further research would be needed to determine whether the PyPrep pipeline includes key features that could benefit our specific implementation.

9 Conclusion

In this project, we set out to develop a pipeline for initial EEG data cleaning and preparation, with a focus on emotion recognition. Our goal was to create a tool that is both accessible and user-friendly, especially for those new to EEG data analysis.

We have successfully built a pipeline capable of processing EEG data and applying initial cleaning methods, including generating visual representations of the data, identifying bad channels, and plotting independent components. The pipeline utilizes established tools and standards, such as the BIDS methodology, PyPrep, and MNE libraries. While our initial aim was to tailor the pipeline specifically for emotion recognition, we realized that a lot is still unknown in this field and further research and analysis needs to be done to be able to achieve this. Therefore, we prioritized retaining as much data as possible, choosing not to include conventional lowpass and highpass filters, which could potentially exclude valuable information for emotion recognition.

The pipeline is designed to be highly user-friendly. Users interact with it through a simple configuration file, where they can set their desired parameters. Many parameters are pre-set, allowing even complete beginners to process their datasets and gain initial insights with minimal effort. By making the pipeline accessible, we aim to support students and researchers interested in EEG but unsure where to begin.

However, the pipeline's functionality remains somewhat limited. While it provides a clear visual overview of EEG data and aids in identifying bad channels or artifacts, further development is needed to make it a comprehensive tool for preprocessing of EEG data. Despite these limitations, we hope this project serves as a stepping stone for those entering the field of EEG research.

10 Acknowledgments

While working on this project we have made use of AI, specifically ChatGPT (version December 2024, OpenAI) for rephrasing and debugging.

11 References

References

[1] P. A. Abhang, B. W. Gawali, and S. C. Mehrotra, Introduction to EEG- and Speech-Based Emotion Recognition, Academic Press, 2016, ch. 1–3. [Online]. ISBN: 9780128044902. Available: https://doi.org/10.1016/B978-0-12-804490-2.00001-4, https://doi.org/10.1016/B978-0-12-804490-2.00002-6, https://doi.org/10.1016/B978-0-12-804490-2.00003-8 [Accessed: Dec. 12, 2024].

[2] A. Gramfort, M. Luessi, E. Larson, D. A. Engemann, D. Strohmeier, C. Brodbeck, R. Goj, M. Jas, T. Brooks, L. Parkkonen, and M. Hämäläinen, "MEG and EEG data analysis with MNE-Python," Frontiers in Neuroscience, vol. 7, p. 267, 2013. doi: 10.3389/fnins.2013.00267. [Online]. Available: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2013.00267/full [Accessed: Dec. 12, 2024].

[3] N. Bigdely-Shamlo, T. Mullen, C. Kothe, K.-M. Su, and K. A. Robbins, "The PREP pipeline: standardized preprocessing for large-scale EEG analysis," Frontiers in Neuroinformatics, vol. 9, Art. no. 16, 2015. [Online]. Available: https://www.frontiersin.org/journals/neuroinformatics/articles/10.3389/fninf.2015.00016 [Accessed: Dec. 12, 2024].

[4] A. Rayi and N. I. Murr, "Electroencephalogram," in StatPearls [Internet], Treasure Island, FL: StatPearls Publishing, updated Oct. 3, 2022. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK563295/ [Accessed: Dec. 12, 2024].

[5] C. S. Nayak and A. C. Anilkumar, "EEG normal waveforms," StatPearls, 2024. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK539805/ [Accessed: Dec. 15, 2024].

[6] J. V. Stone, Independent Component Analysis: A Tutorial Introduction, chapters 1-2 and 11.3, MIT Press, 2004.

[7] E. Larson, "MNE-Python," Zenodo, Aug. 19, 2024. [Online]. Available: https://zenodo.org/records/13340330 [Accessed: Dec. 12, 2024].

[8] Pyprep, "pyprep.Reference," PyPrep Documentation, v0.5.0.dev37+gf50bf36. [Online]. Available: https://pyprep.readthedocs.io/en/latest/generated/pyprep.Reference.html. [Accessed: Dec. 12, 2024].

[9] Pyprep, "pyprep.NoisyChannels," PyPrep Documentation, v0.5.0.dev37+gf50bf36. [Online]. Available: https://pyprep.readthedocs.io/en/latest/generated/pyprep.NoisyChannels.html. [Accessed: Dec. 12, 2024].

[10] K. J. Gorgolewski, T. Auer, V. D. Calhoun, R. C. Craddock, S. Das, E. P. Duff, G. Flandin, S. S. Ghosh, T. Glatard, Y. O. Halchenko, D. A. Handwerker, M. Hanke, D. Keator, X. Li, Z. Michael, C. Maumet, B. N. Nichols, T. E. Nichols, J. Pellman, J. Poline, J. B. Rokem, G. Schaefer, V. Sochat, W. Triplett, J. A. Turner, G. Varoquaux, and R. A. Poldrack, "The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments," Scientific Data, vol. 3, no. 160044, 2016. doi: 10.1038/sdata.2016.44. [Online]. Available: https://www.nature.com/articles/sdata201644 [Accessed: Dec. 12, 2024].

[11] C. R. Pernet, S. Appelhoff, K. J. Gorgolewski, G. Flandin, C. Phillips, A. Delorme, and R. Oostenveld, "EEG-BIDS, an extension to the brain imaging data structure for electroencephalography," Scientific Data, vol. 6, no. 103, 2019. doi: 10.1038/s41597-019-0104-8. [Online]. Available: https://www.nature.com/articles/s41597-019-0104-8 [Accessed: Dec. 13, 2024].

[12] MNE-Python contributors, "Handling bad channels," MNE 1.8.0 documentation, Oct. 24, 2024. [Online]. Available: https://mne.tools/stable/auto_tutorials/preprocessing/15_handling_bad_channels.html [Accessed: Dec. 14, 2024].