Report EEG

Thomas Monninger

# Pipeline Architecture

## Project.py

This file is the starting point for each execution of the pipeline. It defines the sequence of the pipeline and executes the steps in the defined order. A major performance gain is given by parallelization: Calling project.py without specifying a subject will spawn an individual subprocess for each subject and execute the pipeline for all subjects in parallel. Analysis steps require the availability of the results of all subjects, so the logic waits for the subprocess to finish and then calls the analysis steps.

## Config.yaml

The config file separates all configuration parameters. Hence, it is not required to read through the source code. Instead, all relevant parameters can be set centralized in this configuration file.

The configuration file has two sections, one for global settings and one for step-specific settings. Each processing step must have a dict entry in this config file that holds all parameters required to run the specific step.

## Fname

## Base.py

This base class is the core of the pipeline. All processing steps inherit from this class. The class loads the settings from config.yaml. Furthermore, it handles loading of the resulting raw object of the previous pipeline step. This can be achieved by implementing a unidirectional linked list: Each instantiation of Base in the form of a processing step must define the previous processing step. After processing, the Base class takes care of saving the raw object in the form of a fif-file. These fif-files are the only coupling between steps in the processing pipeline. Storing these after each step ensures full transparency and reproducibility of the results.

Most importantly, the base class implements a run() method that generically works for all processing steps:

self.load()

self.process()

self.save()

self.report()

Generally, the methods load(), save() and report() are implemented. When implementing a new processing step, only the process() method needs to be overridden. It defines the step-specific logic.

A major advantage of the Base class is that it can be run standalone. Hence, one does not need to run the full pipeline each step. Instead, during development one can run just one specific step directly from the terminal. Automatically the intermediate results from the previous pipeline step are loaded and made available.

## \_0x\_\*.py

A

## \_1x\_\*.py

## Utils

The utils package is a toolbox of different functions. The idea is to have a wrapper that encapsulates and abstracts functions provided in:

ccs\_eeg\_semesterproject

ccs\_eeg\_utils

# Data

print(raw.info)

    bads: []

    ch\_names: FP1, F3, F7, FC3, C3, C5, P3, P7, P9, PO7, PO3, O1, Oz, Pz, CPz, ...

    chs: 33 EEG

    custom\_ref\_applied: False

    highpass: 0.0 Hz

    line\_freq: 60

    lowpass: 512.0 Hz

    meas\_date: unspecified

    nchan: 33

    projs: []

    sfreq: 1024.0 Hz

    raw.to\_data\_frame().shape

    (33, 683008)

## Event Coding

<https://osf.io/u8w69/>

|  |  |
| --- | --- |
| Event | Code |
| Faces | 1-40 |
| Cars | 41-80 |
| Scrambled Faces | 101-140 |
| Scrambled Cars | 141-180 |
| Response Correct | 201 |
| Response Wrong | 202 |

# Preprocessing Pipeline

The preprocessing pipeline cleans and prepares the data for the subsequent analysis. In this work, preprocessing is performed subject-wise. The steps are shown in figure 1:

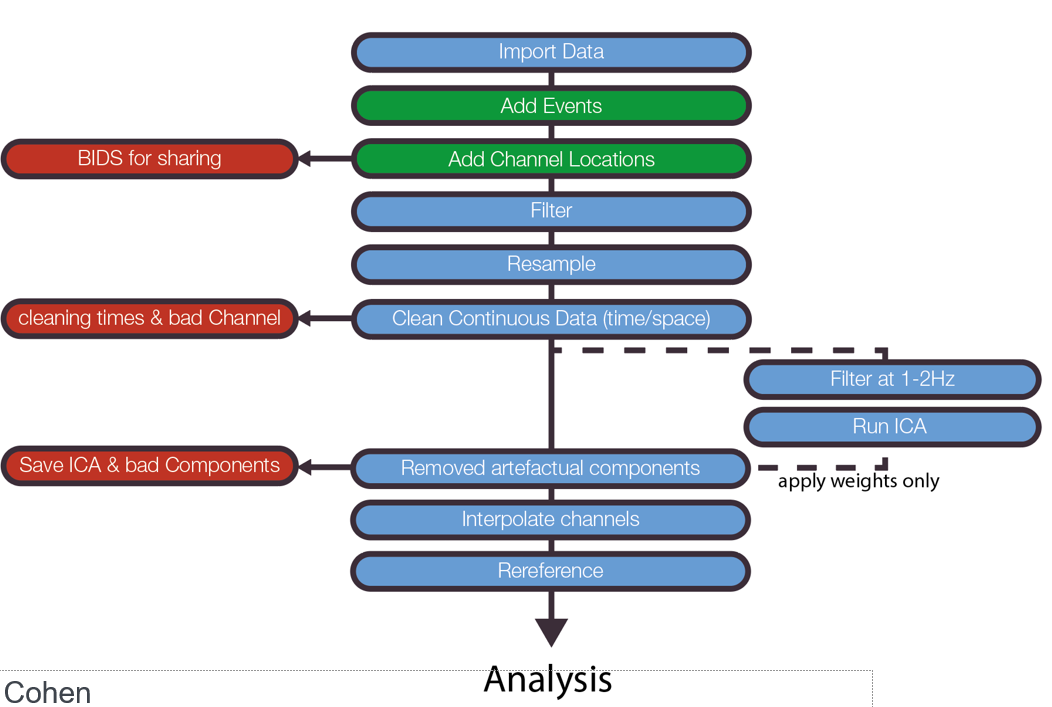


Figure 1: Steps of preprocessing pipeline TODO Ref

As stated in the project requirements, cleaning shall be manually done for three subjects. In this work, subjects 1, 2 and 3 are manually cleaned. These subjects are specified in the config.yaml:

subjects\_preprocess: ["001","002","003"]

At the respective locations in the code a case distinction is performed to either chose the manual cleaning information or load the pre-computed cleaning information:

if self.subject in self.config["subjects\_preprocess"]:

The following subsections will explain the individual steps of the preprocessing pipeline.

## Import Data

To import the N170 dataset, a function from the utils class is used:

utils.load\_data(task, subject\_id)

Under the hood, it reads the data using:

mne.read\_raw\_bids()

The followingRequired because the \*channels.tsv file is not correctly loaded due to problem with naming convention

raw.set\_channel\_types({'HEOG\_left': 'eog', 'HEOG\_right': 'eog',  'VEOG\_lower': 'eog'})

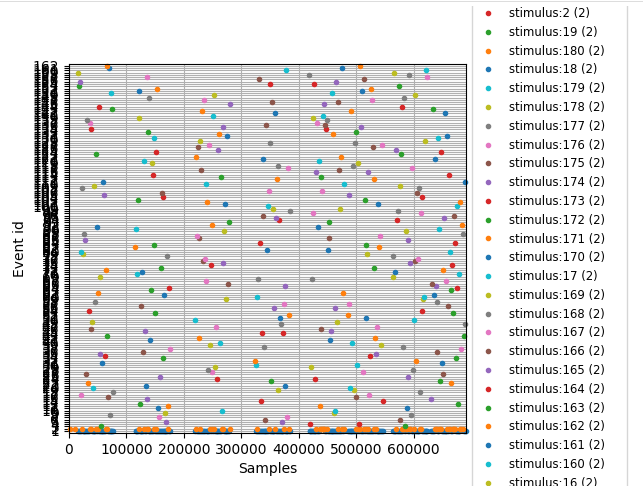
## Add events

The N170 dataset already is in BIDS structure. Hence, the events can just be loaded with the following mne function:

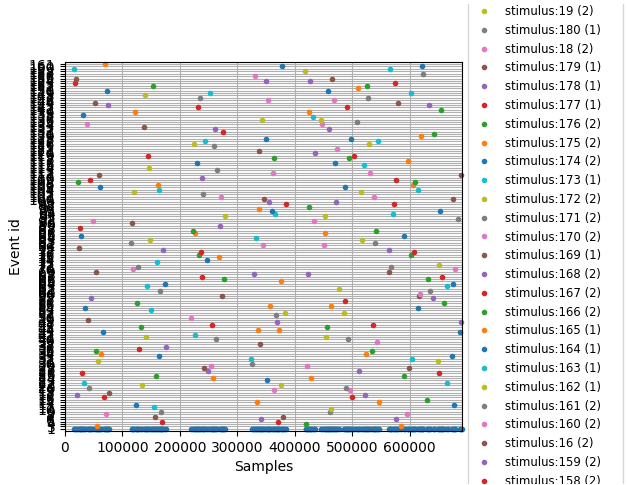
mne.events\_from\_annotations(raw)

As described in the section “Event Coding”, for each trial the subject had to respond. If the response is wrong, i.e., not matching the shown figure in the experiment, it cannot be guaranteed that the subject went through the desired thought process. Hence, data from trials with wrong response is excluded.

Before removing trials with wrong response:



After removing trials with wrong response:



## Add Channel Locations

The N170 dataset consists of 33 channels, that are located according to the so-called “1020” standard montage on the sculp. This is visualized in figure 1.

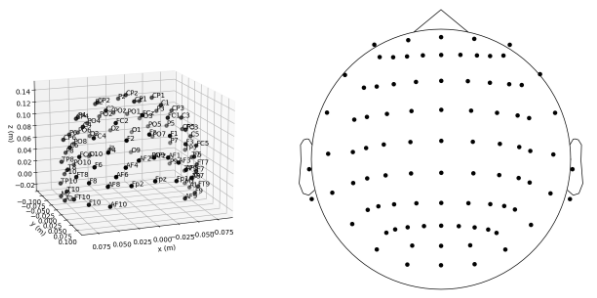


Figure 2: Visualization of 1020 standard montage [1]

To set the channel locations, the following mne function is used:

raw.set\_montage('standard\_1020')

Furthermore, the following three electrodes are available to measure the electrooculogram (EOG), which measures eye movement.

raw.set\_channel\_types({'HEOG\_left': 'eog', 'HEOG\_right': 'eog',  'VEOG\_lower': 'eog'})

## Filter

## Resample

Resampling is optional to reduce the density of measurements in time (i.e., sampling frequency) and therefore increase the processing time. Resampling is not performed here because sufficient compute is available.

## Clean Continuous Data

The raw data might be affected by bad channels or bad time segments. Hence, cleaning is performed as described in the following sections.

### Bad channels

Bad channels usually occur if an electrode is not correctly physically attached to the subject, resulting in no, noisy or drifting signal. An example of a noisy electrode is given in figure 2:



Figure 3: Example of a noisy electrode F8 TODO source

Bad channels can be spotted due to the fact that geometrically adjacent electrodes are strongly correlated and hence have a high inter-channel covariance. In contrast, good channels roughly follow the curve of their neighboring channels. In this work the channels of subject 1, 2 and 3 have been qualitatively evaluated based on the raw data as proposed by the MNE tutorial:

“*Recommended ways to identify bad channels are: […] View raw data with mne.io.Raw.plot() without SSP/ICA enabled and identify bad channels.*” [2]

No strong deviations between the channels have been found for the subjects 1, 2 and 3. This confirms the general impression that the N170 dataset is of high quality.

For all other subjects, the pipeline extracts pre-computed bad channel data. [3] proposes to remove bad channels completely. This is done by adapting the ‘bads’ field of the info object:

raw.info['bads'].extend(bad\_channels)

The MNE framework by default applies the following to exclude channels marked this way:

exclude='bads'

Finally, bad channels are interpolated as described in [3] with the following method:

raw.interpolate\_bads()

### Bad segments

## Remove Artifactual Components using ICA

The Independent Component Analysis (ICA) is performed to decompose a mixed signal into its independent subcomponents. This way, undesired artefactual EEG components like muscle noise can be excluded from the signal.

* Perform ICA on epochs to see, whether component is undesired or occurring in each epoch
* ICA components are sorted by relevance to the signal, hence focus on the initial few components

### Pre-ICA Filtering

“ICA is sensitive to low-frequency drifts and therefore requires the data to be high-pass filtered prior to fitting. Typically, a cutoff frequency of 1 Hz is recommended.” [5]

For this reason, a separate filter is applied to a copy of the raw object before extracting the independent components. A high-pass filter is applied with a lower transition bandwidth of 2.0 Hz resulting in a -6dB cutoff frequency of 1.0Hz.

FIR filter parameters

* Designing a one-pass, zero-phase, non-causal highpass filter:
* Windowed time-domain design (firwin) method
* Hamming window with 0.0194 passband ripple and 53 dB stopband attenuation
* Lower passband edge: 2.00
* Lower transition bandwidth: 2.00 Hz (-6 dB cutoff frequency: 1.00 Hz)
* Filter length: 1691 samples (1.651 sec)

### Manual evaluation of components

### Apply ICA to exclude artifacts

ica.apply(self.raw, exclude=bad\_comps)

# Analysis

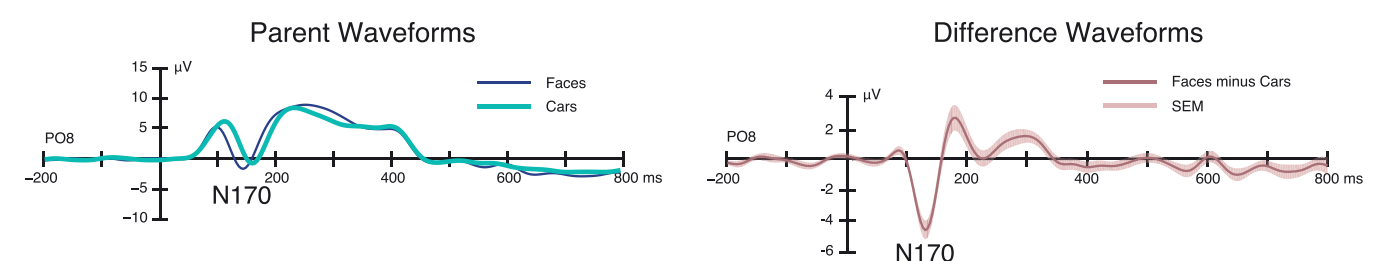
## ERP Peak Analysis

Extract the study-relevant ERP peak subjectwise (e.g. one value per subject) and statistically test them.

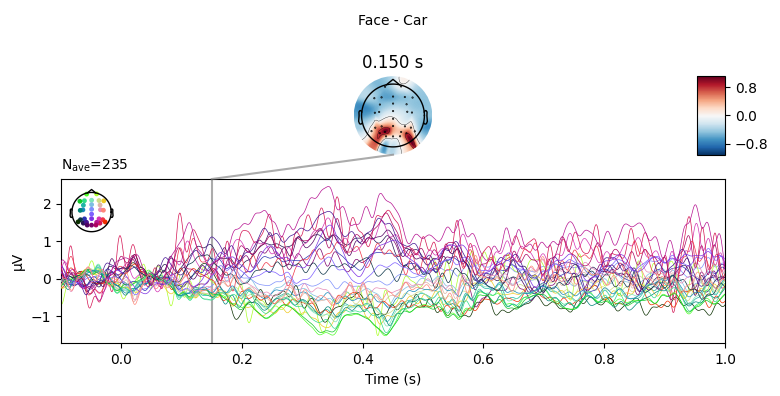
RQ: On which ERP-peaks do we find major difference between the conditions?

Channels, times, peaks for N170: --> Rossion 2008, ERP Core Paper

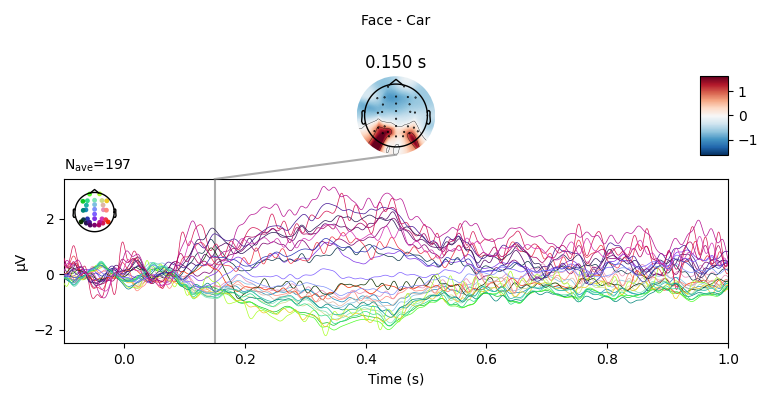
P7, PO7, P8, PO8; 130-200ms after stimulus onset



Before removing trials with wrong response:



After removing trials with wrong response:



# References

[1] *Working with sensor locations — MNE 0.22.0 documentation*. (n.d.). Retrieved March 28, 2021, from <https://mne.tools/stable/auto_tutorials/intro/plot_40_sensor_locations.html>

[2] *Rejecting bad data (channels and segments) — MNE 0.15 documentation*. (n.d.). Retrieved March 28, 2021, from <https://mne.tools/0.15/auto_tutorials/plot_artifacts_correction_rejection.html>

[3] *Interpolating bad channels — MNE 0.22.0 documentation*. (n.d.). Retrieved March 28, 2021, from <https://mne.tools/stable/auto_tutorials/preprocessing/plot_15_handling_bad_channels.html>

[4] Makeig, S., Bell, A. J., Jung, T. P., & Sejnowski, T. J. (1996). Independent component analysis of electroencephalographic data. *Advances in neural information processing systems*, 145-151.

[5] *mne.preprocessing.ICA — MNE 0.22.0 documentation*. (n.d.). Retrieved March 28, 2021, from <https://mne.tools/stable/generated/mne.preprocessing.ICA.html>