EEG Preprocessing and analysis [WS2020]

Project report by Thomas Monninger, Matrikelnummer 3470145 31.03.2021

Abstract

This report discusses the work performed on the N170 dataset: a face-viewing experiment with an effect at 170ms. Integral part is the pipeline architecture, which is introduced in chapter 1. Chapter 2 discusses the data structure and properties. In section 3, the preprocessing pipeline is explained with all the required steps. The final chapter 4 presents the analysis that has been performed on the preprocessed data. First, an ERP peak analysis was carried out including statistical testing. The other two analysis parts were chosen out of a list of four. One is a decoding analysis to decode the main contrast of the experiment across time and statistically test at which points in time is information about the conditions available. The last step is a time frequency analysis to calculate underlying oscillations per effect and check statistical significance.

Table of contents

1 Pipeline Architecture

Central element of the project is the source code. The source code is written in a modular and clean manner and serves as additional documentation. Also, the actual parameters and functions that have been used are all manifested there. This report aims to give an overview about the most relevant details and discusses results. However, it shall be evaluated in conjunction with the code. Hence, the report is integrated into the repository.

The following subsections explain the pipeline architecture.

1.1 Project file

Files: 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.

1.2 Configuration file

Files: 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.

1.3 Fname

Files: fnames.py, config.py

The methodology of this work strongly follows the paper from van Vliet [2]. The goal is to analyze multiple subjects in a fully reproducible and documented way. Hence, his 7 proposed tips are applied here:

1. Every analysis step is a separate script
2. A script processes either subject-specific data or aggregates across subjects (here: explicitly indicated by enumeration convention of the scripts)
3. One master script runs the entire analysis (here: project.py)
4. All intermediate results are stored (here: the base class serves as template and automatically loads persisted data from the previous processing step and stores the changed data at the end of the script)
5. Visualize all intermediate results (here: each script creates figures that are added to an MNE report; one per subject and one for the aggregated information)
6. Each parameter and file name is defined only one (here: all parameters are centralized in config.yaml)
7. Distinguish pipeline from other scripts (here: pipeline scripts have leading number)

Furthermore, the logic of “fname” is copied from van Vliet’s repository “conpy”. The code is under BSD 3-Clause License and hence can be used for this project. Still, all rights belong to the author, which is also credited in the source code files of “fnames.py” and “config.py”. The “fname” logic handles paths in a structured and centralized way and further improves reproducibility.

1.4 Base class template

Files: 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.

1.4.1 Preprocessing steps per subject

Files: _0x_*.py

In line with the seven tips of van Vliet [2], all files that do preprocessing per subject have the suffix “_0”.

1.4.2 Pre-analysis steps per subject

Files: _1x_*.py

In line with the seven tips of van Vliet [2], all files that do pre-analysis per subject have the suffix “_1”.

1.4.3 Analysis steps across subjects

Files: _2x_*.py

In line with the seven tips of van Vliet [2], all files that do analysis across subjects have the suffix “_2”.

1.5 Utils

Files: utils.py, ccs_eeg_semesterproject.py, ccs_eeg_utils.py

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
```

Furthermore, the package contains useful functions which are used across the pipeline.

2 Data

The data of the N170 dataset contains experiments with 40 subjects. For each subject, the data includes 30 electroencephalography (EEG) channels, which measure the electrical activity of the brain, and 3 electrooculogram (EOG) channels, which measures eye movement. The data is sampled with 1024Hz and the line frequency is 60 Hz.

```
print(raw.info)
    bads: []
    ch_names: FP1, F3, F7, FC3, C3, C5, P3, P7, P9, PO7, PO3, O1, Oz, Pz, ...
    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
```

The data from an example subject 001 has the following shape:

```
raw.to_data_frame().shape
 (33, 683008)
```

Given a sampling rate of 1024Hz, the time length of the experiment in this case is:

$$\frac{683008}{1024\frac{1}{s}} = 667s \approx 11min$$

2.1 Event Coding

In the N170 dataset the subjects were monitored while perceiving different events. Furthermore, they had to react to the displayed event by pressing the corresponding button. The correctness of the answer is also provided.

The event coding of the N170 project is as follows [14]:

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

2.2 Folder structure

The repository “eeg_project” is hosted on GitHub and has the following folder structure:

```
eeg_project
├── annotations   → Manual cleaning information for 3 subjects
├── doc           → Documentation, e.g., this report
├── reports       → HTML reports generated by pipeline
├── results       → Output of across-subject processing steps
├── subjects      → Output of per-subject processing steps
├── files         → Files described within this report
```

3 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 following the proposal of Cohen [22] as shown in figure 1. The order matches what Luck proposes in his book “An Introduction to the Event-Related Potential Technique” [15].

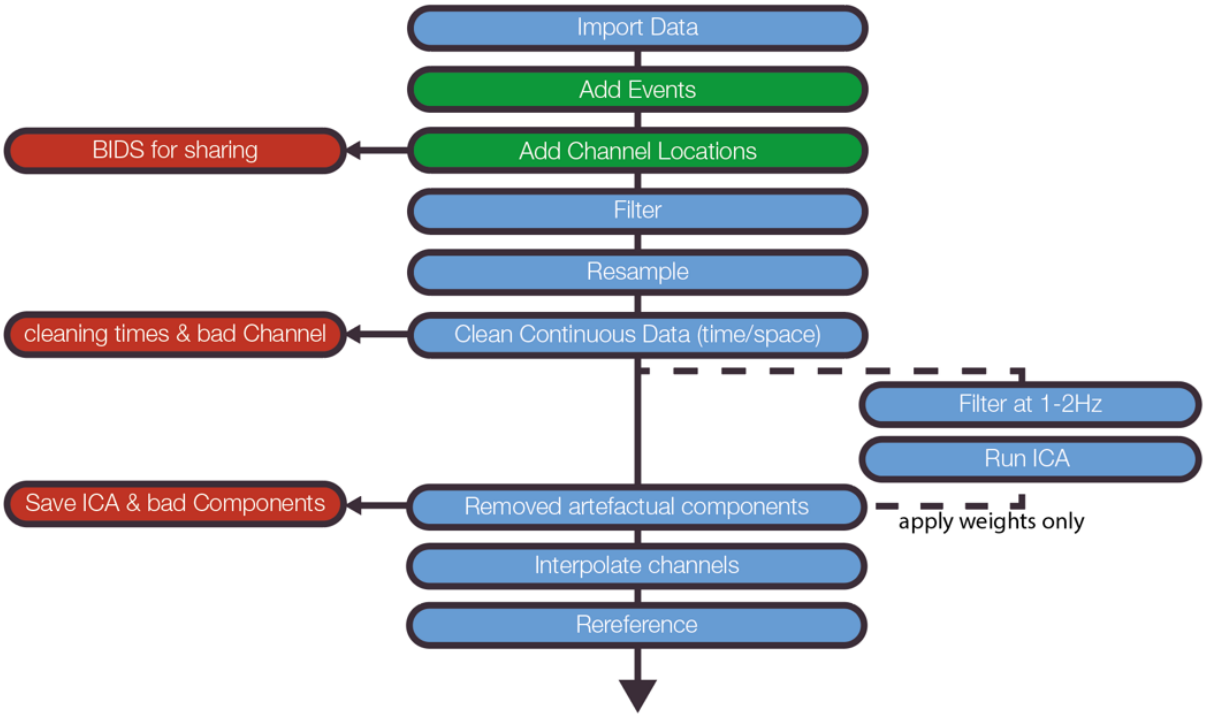


Figure 1: Steps of preprocessing pipeline as proposed by Cohen [22]

As stated in the project requirements, cleaning shall be manually done for three subjects. In this work, subjects 002, 003 and 004 are manually cleaned and specified in the config.yaml:

```
subjects_preprocess: ["002","003","004"]
```

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. Please be aware that for each subject a report is

3.1 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 following snippet is required because the *channels.tsv file is not correctly loaded due to problem with naming convention in MNE:

```
raw.set_channel_types({'HEOG_left': 'eog', 'HEOG_right': 'eog',  'VEOG_lower':
 'eog'})
```

3.2 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. The overview of event IDs is shown in figure 2. The upper plot shows many orange dots at event ID = 2, which represent events with wrong responses. The lower plot does not contain these anymore after cleaning.

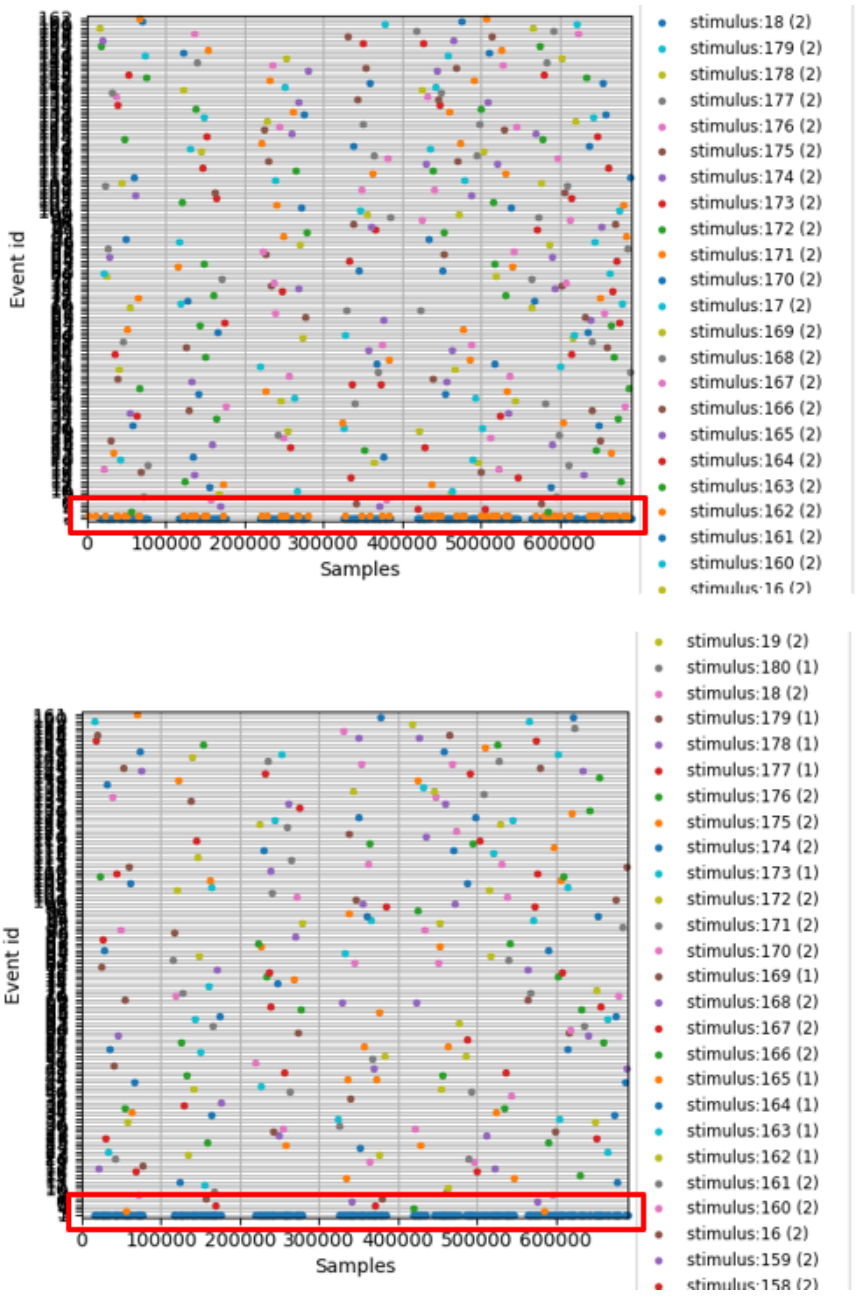


Figure 2: Overview over distribution of events sorted by event id. Event ID 2 corresponds to wrong response (upper in orange) and is not available anymore after removing (lower).

3.3 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 3.

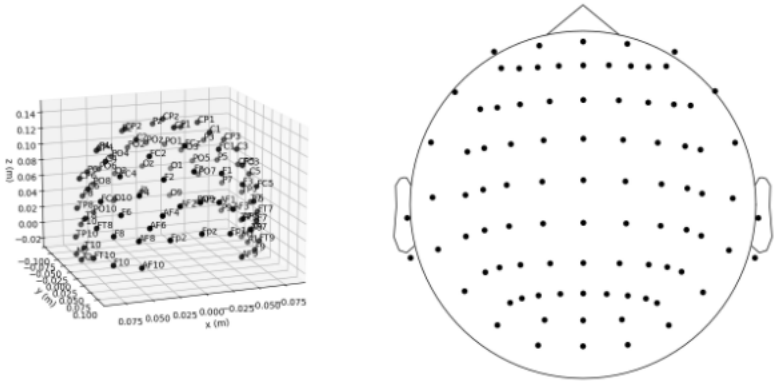


Figure 3: Visualization of 1020 standard montage [17]

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'})
```

3.4 Filter

Files: _00_filter.py

Filtering is applied to remove noise in low and high frequency bands. Activity in low frequency bands includes undesired systematic drifts of potential, and activity in high frequency bands can include noise, for example introduced by power line frequency [9].

3.4.1 High-pass Filtering

Literature states that using $f_{cutoff} > 0.1$ Hz can introduce a systematic bias [10, 11]. This might affect interpretation of neural activity especially in event-related potential (ERP) analysis, as done in this work. A lower bandpass edge of 0.45 Hz is chosen resulting in a -6dB cutoff frequency of 0.23 Hz. This turned out to be the best trade-off between loosing information for ERP analysis and removing undesired potential drifts based on empirical inspection of the filtered result.

3.4.2 Low-pass Filtering

An upper bandpass edge of 50 Hz is chosen resulting in a -6dB cutoff frequency of 56.25 Hz. This way, the line noise of 60 Hz and other artefacts in higher frequency bands are effectively removed with minimizing the loss of actual signal.

Additionally, [9] gives a recommendation to low-pass filter the raw data at $\frac{1}{3}$ of the desired sample rate in order to avoid reduction in temporal precision during resampling. As clarified in the

subsequent section “Resample”, no resampling is done and hence no additional action was performed.

VanRullen stated that low-pass filtering using non-causal filters might shift activity to earlier or later times compared to when it truly happened [12]. Though later publications could delimit this finding, it is being kept in mind during this project.

3.4.3 Implementation

As a filter type, a finite impulse response (FIR) filter is chosen. This choice is guided by Widmann et al. [10], where they propose using FIR filters over IIR filters for most purposes in electrophysiological data analysis. The actual filter follows a windowed FIR design and sticks to the default in EEG analysis [8]. A bandpass filter is used to simultaneously filter low-pass and high-pass. In this work, the list of filter parameters is provided as proposed in [10]:

FIR filter: one-pass, zero-phase, non-causal bandpass filter:

- Windowed time-domain design (firwin) method
- Hamming window with 0.0194 passband ripple and 53 dB stopband attenuation
- Lower passband edge: 0.45 Hz
- Lower transition bandwidth: 0.45 Hz (-6 dB cutoff frequency: 0.23 Hz)
- Upper passband edge: 50.00 Hz
- Upper transition bandwidth: 12.50 Hz (-6 dB cutoff frequency: 56.25 Hz)
- Filter length: 7511 samples (7.335 sec)

An exemplary result of the chosen filter parameters can be seen in figure 4 for subject 040. The frequency scale is logarithmic, the ordinate displays the signal power in $\frac{\mu V^2}{Hz}$. The decibel scale is used to indicate the ratio between the electric potential and frequency on a logarithmic scale. The upper plot shows the power spectral density before filtering. As typical for natural signals, the power decreases for increasing frequency. The line at 60Hz overlaps with a peak in the signal power. This comes from power line noise. The EEG data has been captured in California, where 60Hz is the default line frequency.

The lower plot indicates the power spectral density after filtering. The effect of the high-pass filter is not visible, since the affected frequency band is not plotted. An additional line at 50Hz indicates the upper bandpass edge of the low-pass component of the chosen bandpass filter. From this point on, the power decreases rapidly with increasing frequency. The peak at 60Hz remains very small after filtering, as desired during conception. The -6dB cutoff frequency of a low-pass filter with upper passband edge of 50Hz is at 56.25Hz, meaning that at 56.25Hz the signal power is only $\frac{1}{4}$ of the original signal. At higher frequencies, the signal continues to be damped exponentially in relation to the frequency, so high frequencies are effectively cut.

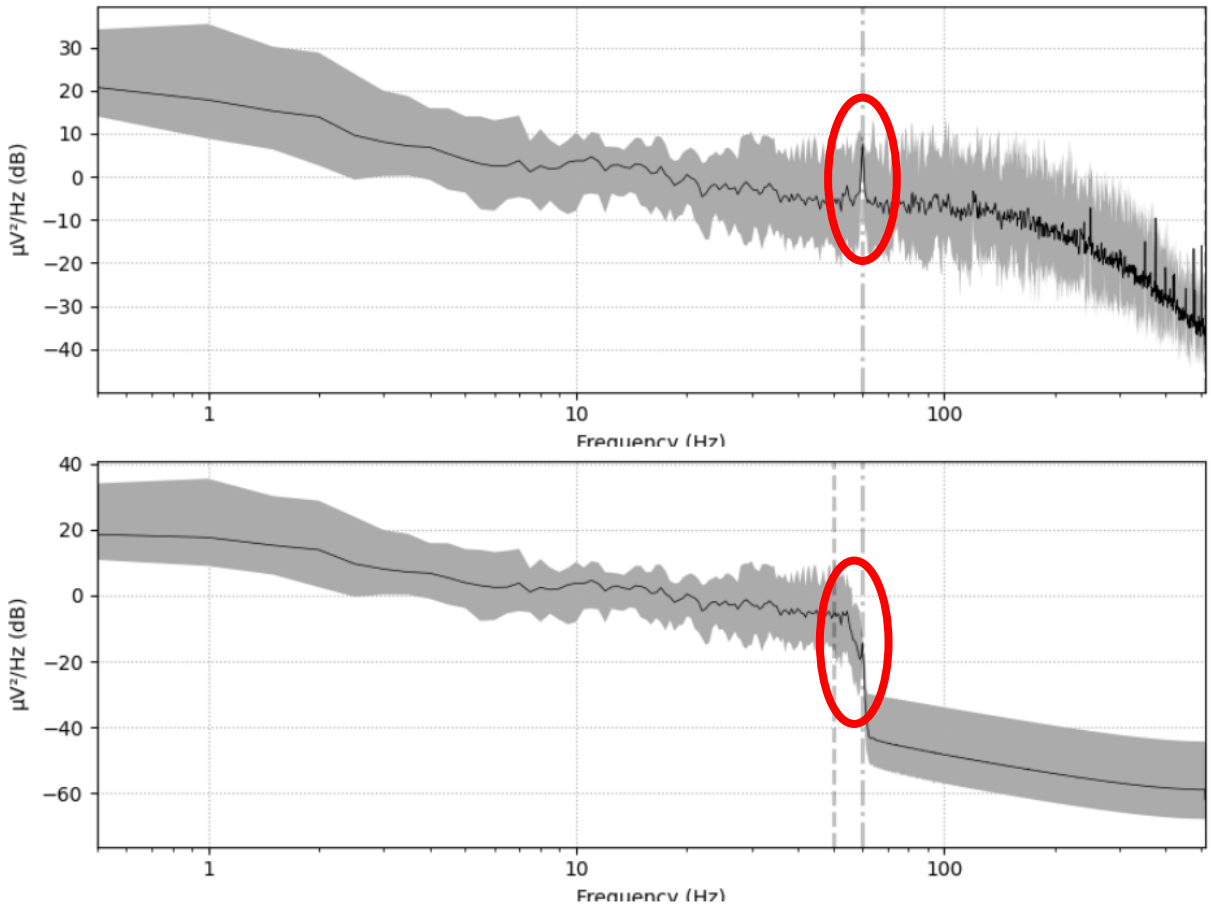


Figure 4: Power spectral density diagram before (upper) and after (lower) bandpass filtering subject 040.

3.5 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.

3.6 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.

3.6.1 Bad channels

Files: _01_clean_channels.py

Bad channels usually occur if an electrode is not correctly physically attached to the subject, resulting in no, noisy or drifting signal. 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 002, 003 and 004 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.*" [4]

No strong deviations between the channels have been found for the subjects 002, 003 and 004. 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. [5] 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'
```

3.6.2 Bad segments

Files: _02_clean_segments.py

Specific time segments of the experiment might be corrupted due to various reasons. One example is body motion of the subject under study, which results in strong artefacts across multiple channels.

The MNE plot tool allows to interactively mark bad segments. For the three manually cleaned subjects "002", "003" and "004", the procedure is as follows: First, a new label has been created, in this case "BAD_segment". Then, the potentials were checked during the full experiment and bad segments were marked. Those markings are stored in the dict raw.annotations. Finally, the markings were saved to the folder /annotations:

```
self.raw.annotations[bad_ix].save(path_annotations)
```

Unfortunately, the proposed way using CSV export did not work with MNE in version 0.22.0. Exporting markings to CSV file filled the onset with an invalid timestamp and the duration value was lost. Hence, in this work the export and import is done with txt files.

Figure 6 shows an example of cleaning bad segments of subject 002 with the GUI of MNE.

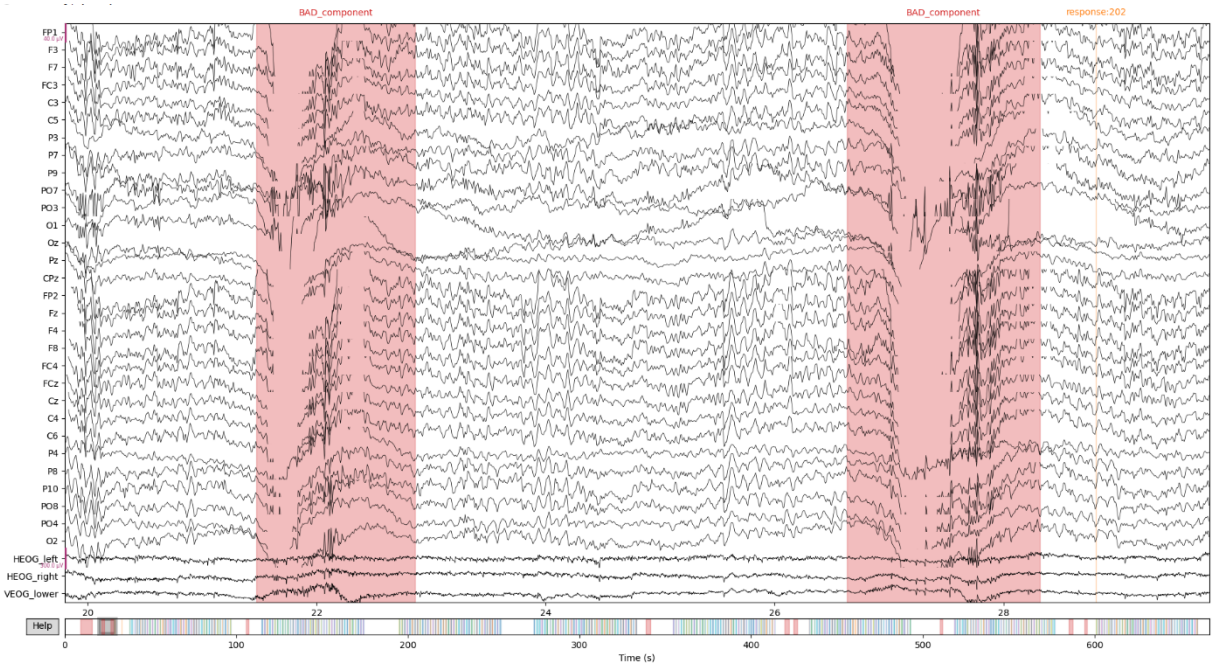


Figure 5: Example of manually cleaning bad segments of subject 002 with the MNE GUI.

During processing, a comparison plot is generated for each of the subjects. Figure 7 shows the results at electrode PO8 of subject 040. The resulting signal of all 4 cleaning methods is visualized: raw (uncleaned), manual cleaning (using either the pre-computed or the manual cleaning annotations), cleaning by threshold and the use of AutoReject. It can be clearly seen that manual cleaning and AutoReject are close to the raw signal, which means that only few bad segments were present in the signal. Threshold cleaning gives a quite different result. When sanity-checking all subjects, this was a general observation. Indication is given that threshold cleaning does not work sufficiently well and should not be used. On the other side, AutoReject and the data provided for manual cleaning are very similar for most of the subjects. One could hence argue that the AutoReject algorithm is pretty sophisticated and the manual cleaning procedure produced no obvious errors. Also, for the three manually cleaned subjects the bad segments have been cross-checked with AutoReject and generally a good overlap has been identified.

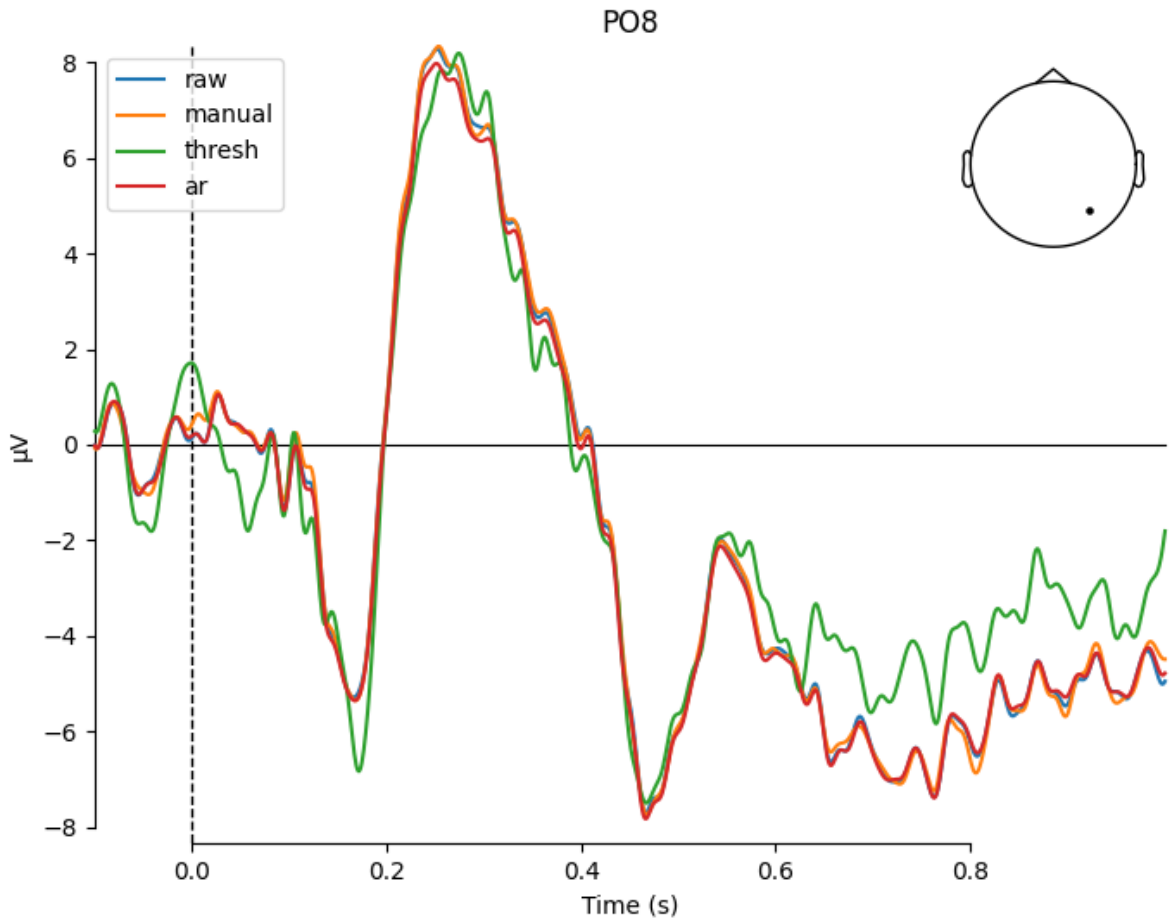


Figure 6: Comparison plot between different methods of cleaning bad segments. Electrode PO8 of Subject 040 is shown.

3.6.3 Bad subjects

Files: config.yaml

Filtering bad subjects is usually not part of the regular preprocessing and cleaning pipeline. However, during this project, extensive sanity-checks have been performed after each step for each subject. Already in early preprocessing steps, indications have been recognized regarding the low quality of subject 001. A clear picture was given when looking at the ERP plot of the difference signal between

the conditions “faces” and “cars”, carried out in chapter “ERP Peak Analysis”. This plot is shown in figure 8 on the left. As a comparison, the same visualization is shown on the right for subject 002. It is a lot less noisy and generally representative for the quality that can be expected from a subject of good quality.

The rejection of subject 001 matches the procedure of the ERP core paper [1]. In their provided subject summary, subject 001 is rejected due to “less than 50% faces trials remaining after artifact rejection” [23].

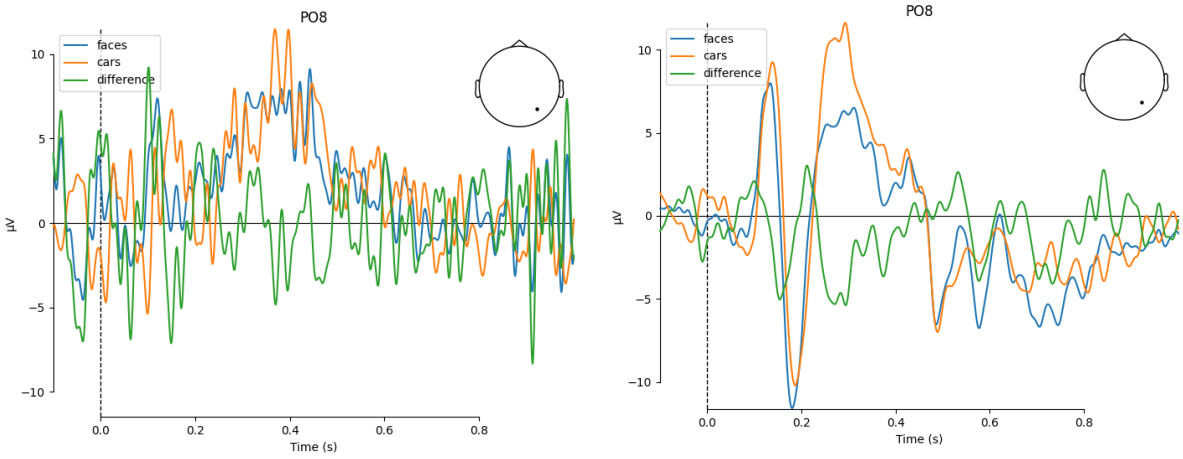


Figure 7: ERP plot of difference between conditions "faces" and "cars". Subject 001 is plotted on the left and subject 002 is plotted on the right. It is clearly visible how noisy the signal of subject 001 overall looks, especially in comparison.

Another bad subject has been identified when sanity-checking intermediate results at a later stage. During time-frequency analysis one activation didn’t match the expected outcome. Subject 029 was identified to cause this activity. A more detailed look into the preprocessing figures revealed an incorrect epoch. As shown in figure 9, subject 029 has extremely high activation in epoch 48. This anomaly yields electric potential with more than 300 µV, which is roughly factor 10 of usual maximum activity. The result was a strong bias even visible in the evoked component. At $t = 0s$, the evoked potential is more than 5 µV and hence much larger than expected. This is very well visible in figure 10.

As a remedy, multiple options were possible: O the one side, the false epoch could be marked, removed and interpolated. However, this subject was not within the three subjects, that are cleaned manually in this work. Much more important is that the root cause for this wrong epoch is not clear. Hence, other epochs might be affected as well. The more conservative and safe way was to remove this subject from the list of processed subjects. This is what has been finally done in this work.

During sanity-checking, few other subjects have been found that have a signal which does not fully match the expected behavior. In contrast to subjects 001 and 029, they do not vary as extremely from a qualitative point of view. Also, some variation is in the nature of electrophysiological data and it is really important to not simply remove all subjects, which do not match the expectations. This type of cherry-picking data is an anti-pattern of data processing and it is a general rule of statistics to only remove data if there are strong arguments in place to do so.

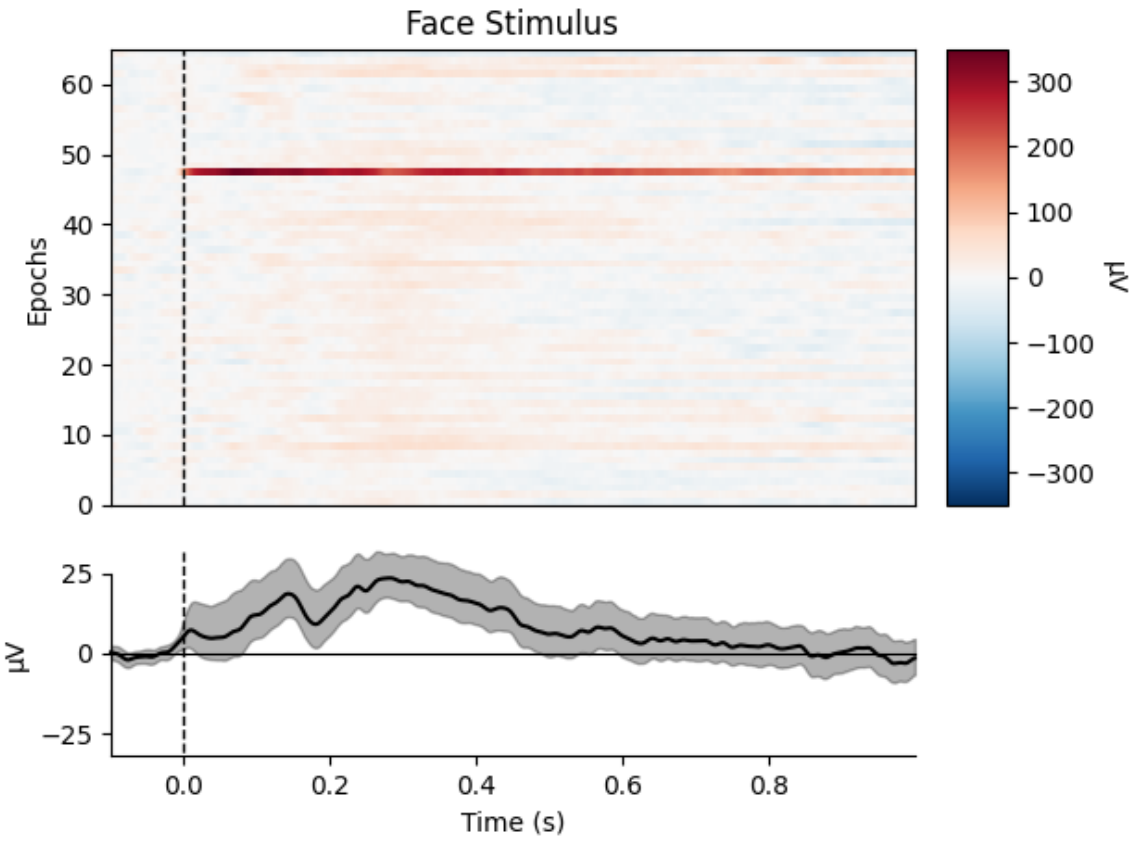


Figure 8: Activity across epochs for subject 029 considering face stimulus. It can be clearly seen that epoch 48 carries an extreme outlier signal, reaching 300 µV.

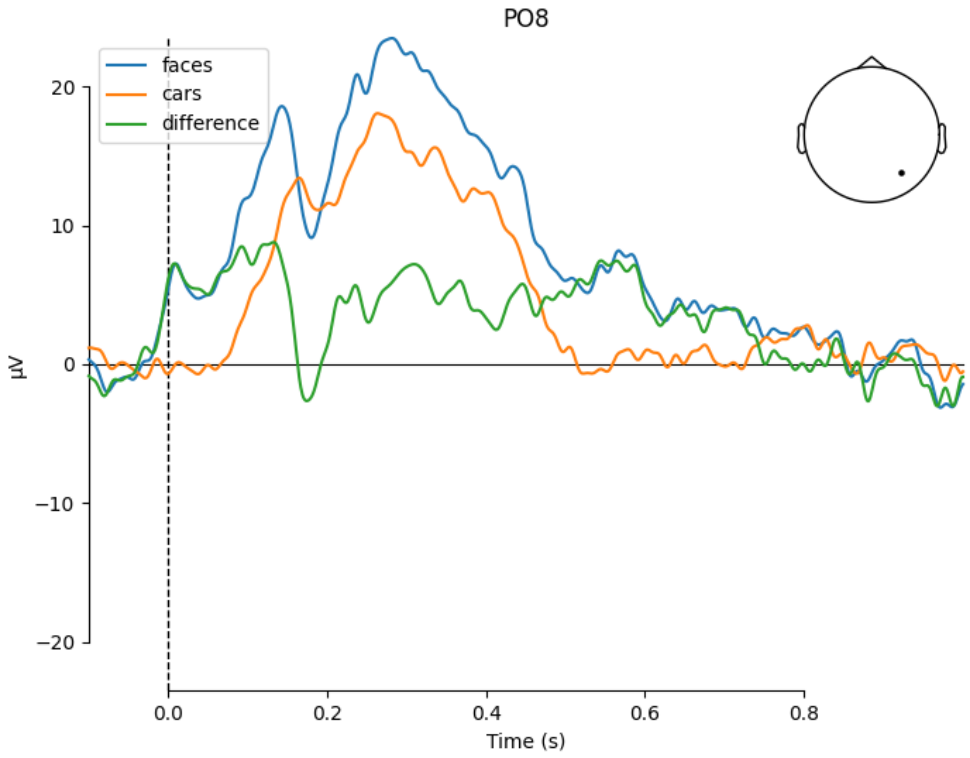


Figure 9: A strong drift of the curve for condition "faces" is detected for Subject 029 before t=0s.

In the previous paragraph two subjects have been identified that shall be excluded from the analysis. For this case, the pipeline offers a dedicated entry in the config.yaml file:

```
# Subjects considered in analysis
subjects: ["002","003",[…], 039","040"]
```

Excluding specific subjects is straight-forward with the given pipeline. One just needs to remove the corresponding subject IDs from the list in the configuration file. At the next run of the pipeline, only the remaining subjects are considered in the analysis.

3.6.4 Baseline correction

Files: _02_clean_segments.py

Baseline correction is essential to remove the systematic offset of the electric potential. It is typically applied when the data is epoched [15]. In this work, the automated baseline correction of the MNE toolbox is used. According to the source code, baseline correction is applied at instantiation of the mne.Epochs object. It is calculated individually per epoch and channel by subtracting the mean of the baseline period from the entire epoch [16]. As shown in the exercise, the baseline might be dependent on the condition. In order to avoid a systematic offset, the baseline is corrected individually per condition in this work using the described built-in MNE functionality. It is kept in mind during the analysis that the choice towards not correcting the general baseline might have negative influence.

The baseline period is chosen equally to the default setting:

```
baseline = (None, 0)  # from the first instance to t = 0
```

In this work, epoching is done with the following time limits as defined in config.yaml:

```
epoch_tmin: -0.1  # s
epoch_tmax: 1.0  # s
```

Hence, the baseline period is [-0.1s, 0.0s].

3.7 Remove Artifactual Components using ICA

Files: _03_ica.py

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 [3]. Two tricks are applied to ease the identification of artifactual components:

- 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

A disclaimer in advance: The calculation of ICA components apparently is non-deterministic. This has been found out only at the very end of the project. Unfortunately, no random seed was set when

creating the overview of manual cleaning ICA artifacts. Therefore, another execution might yield slightly different results.

3.7.1 Pre-ICA Filtering

The MNE tutorial gives the following recommendation [5]: "*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.*" 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 the proposed -6dB cutoff frequency of 1.0Hz. For the same reasoning as in the chapter "Filter", a windowed finite impulse response filter is used and the list of filter parameters is provided as proposed in [10]:

FIR filter parameters: 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 Hz
- Lower transition bandwidth: 2.00 Hz (-6 dB cutoff frequency: 1.00 Hz)
- Filter length: 1691 samples (1.651 sec)

3.7.2 Manual evaluation of components

In the following, the manual identification of artifactual ICA components is shown for three subjects, 002, 003 and 004. The property plots have been evaluated per ICA component in order to identify undesired artefacts produced by eye, muscle, heart or other sources of noise. The guideline of the University of California San Diego has been applied [25].

3.7.2.1 Subject 002

An overview of the ICA components of subject 002 is shown in figure 11. Figure 12 shows the property plots for ICA components 002 and 013. Both are considered as artefacts for the following reasons:

- 002: General eye artefact: ECDs near eyes, power concentrated at low frequencies
- 013: Horizontal eye movement: ECDs near eyes, power concentrated at low frequencies, power spectrum slightly looks like step function

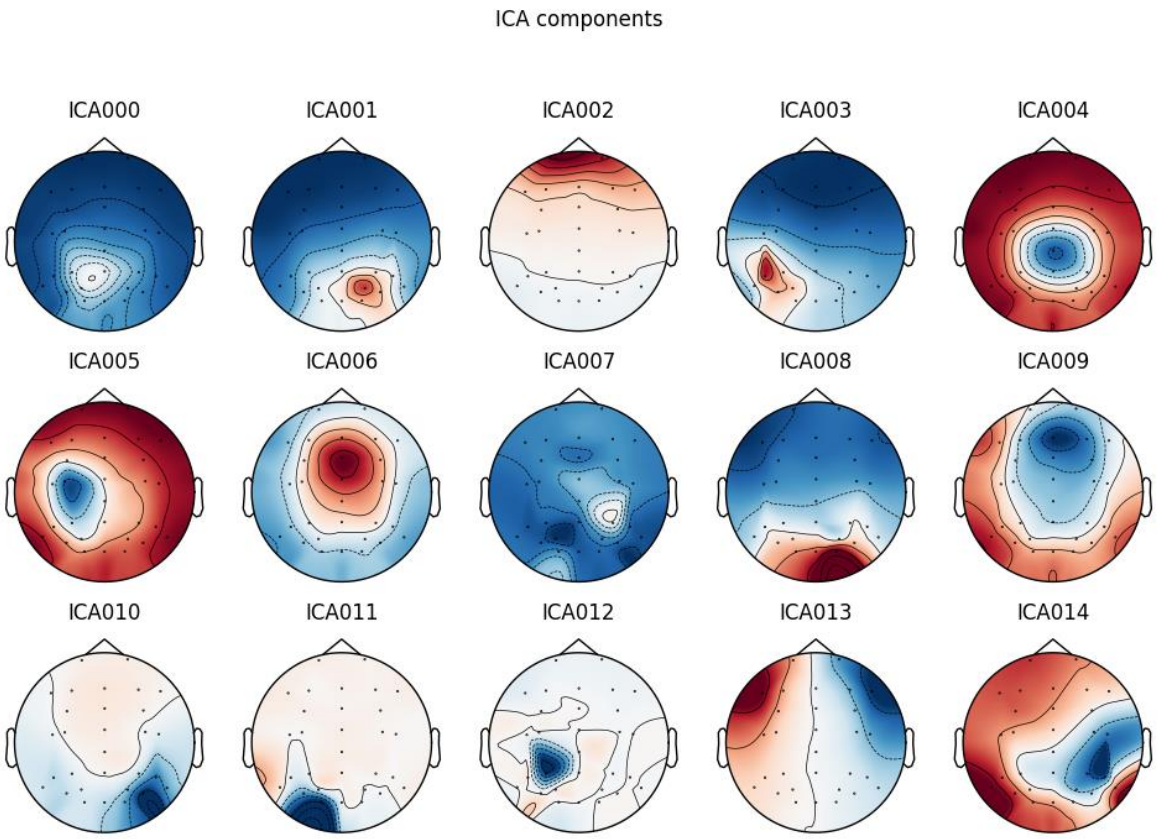


Figure 10: ICA components of subject 002.

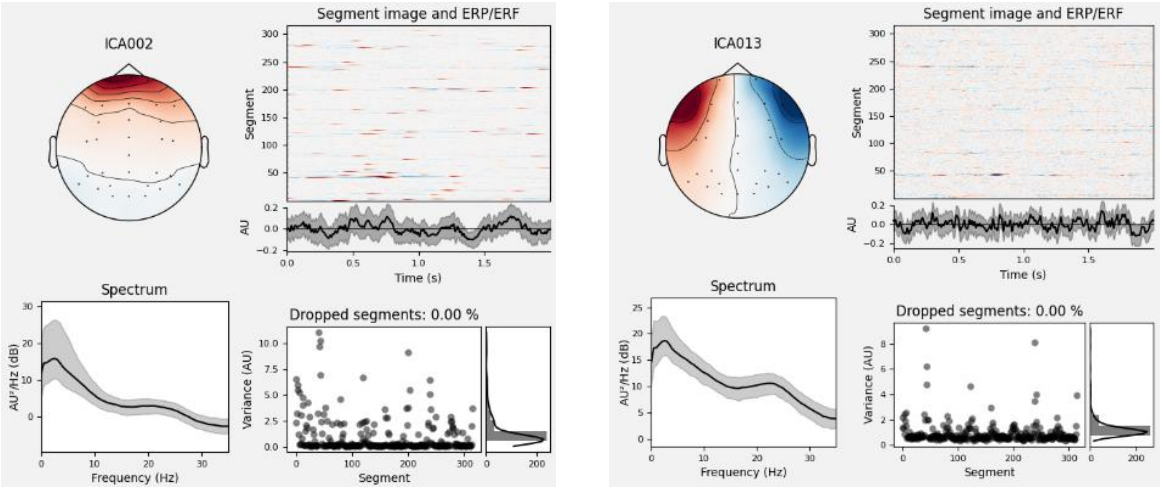


Figure 11: Properties of ICA components 002 and 013 of subject 002.

The results of removing artifactual components of the ICA are checked with an ICA overlay plot. This epoched view is shown in figure 13. The original signal before removing ICA artifacts is drawn in red and the result is overlayed in black. It can be clearly seen that the big outliers are not anymore part

of the signal after removing artifacts. Hence, it can be concluded that the artifact removal was successful.

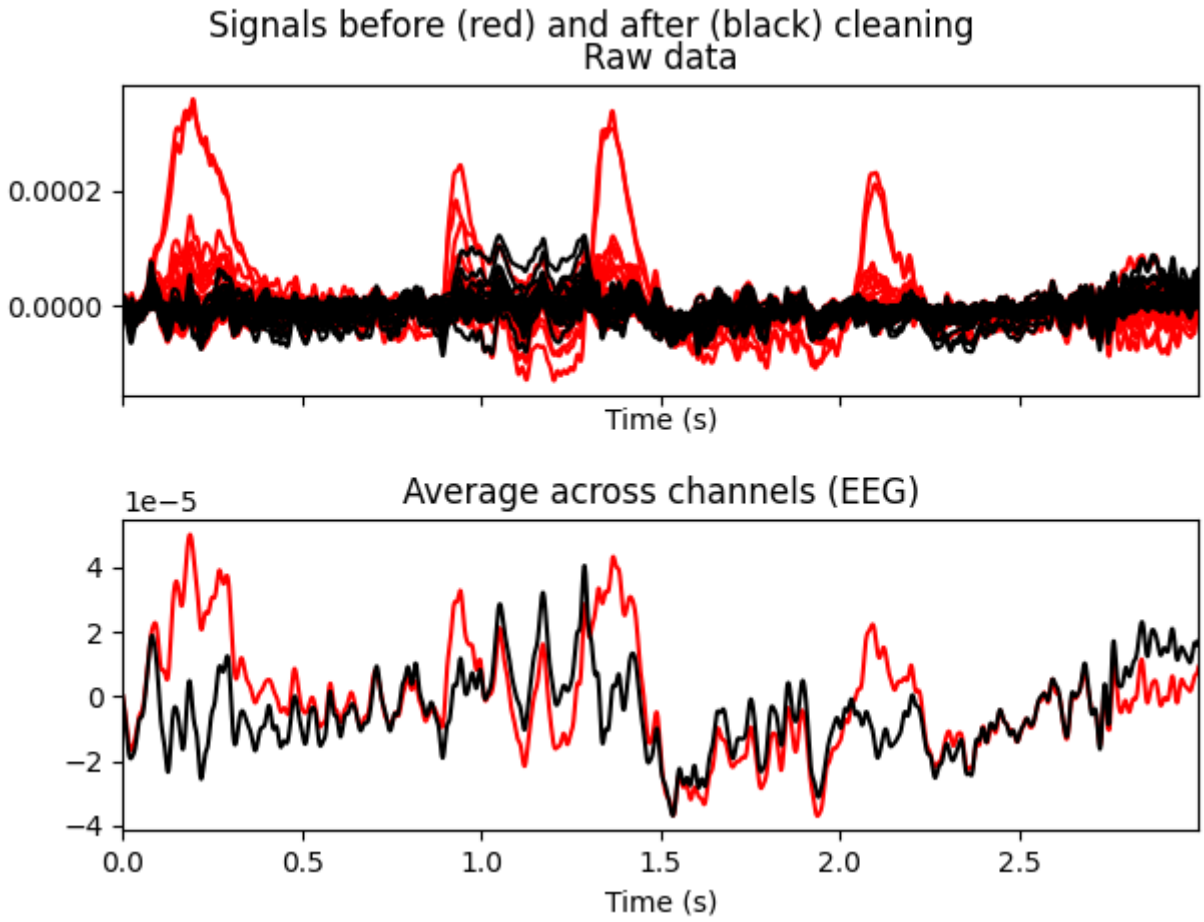


Figure 12: ICA overlay plot of subject 002, signal before cleaning (red) and after cleaning (black) is overlayed.

3.7.2.2 Subject 003

An overview of the ICA components of subject 003 is shown in figure 14. Figure 15 shows the property plots for ICA components 000, 008 and 009. Those are considered as artefacts for the following reasons:

- 000: General eye artefact: ECDs near eyes, power concentrated at low frequencies
- 008: General eye artefact: ECDs near eyes, power concentrated at low frequencies
- 009: Muscle artefact: Peak at 20Hz (or potentially also eye artefact)

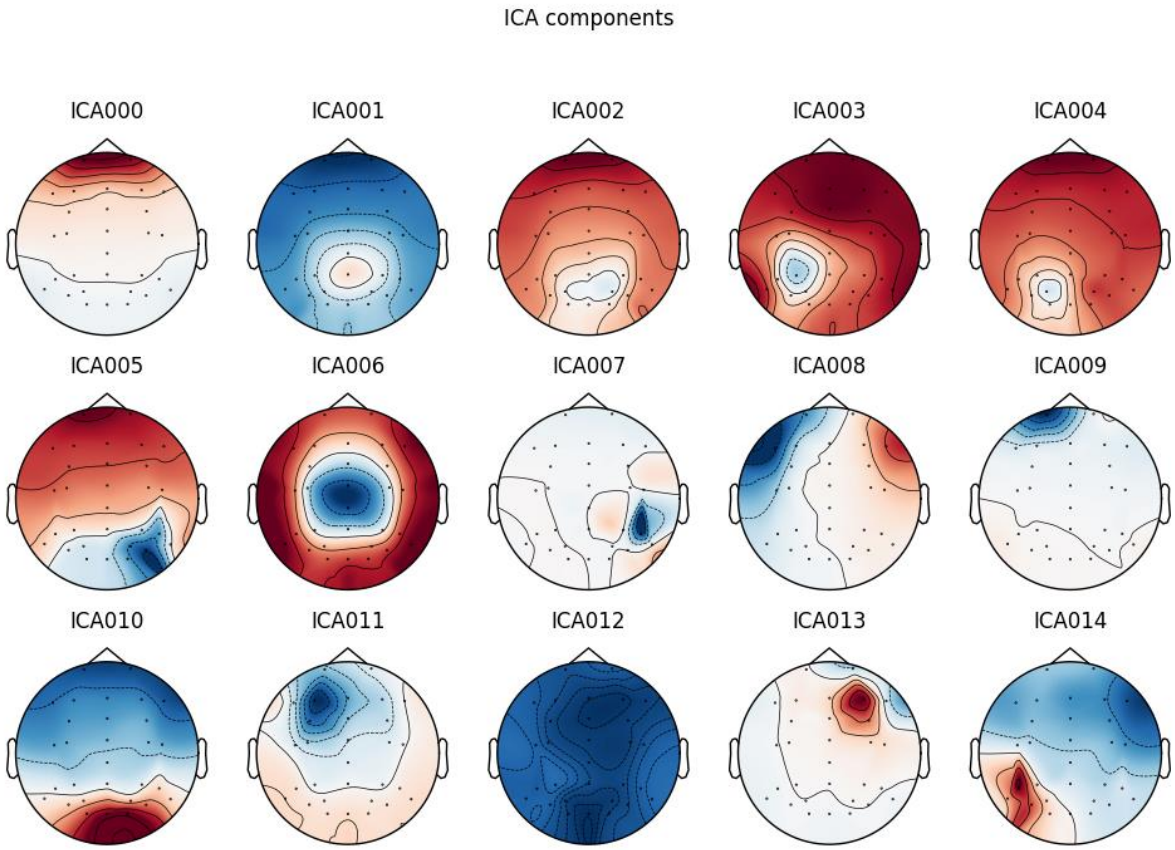


Figure 13: ICA components of subject 003.

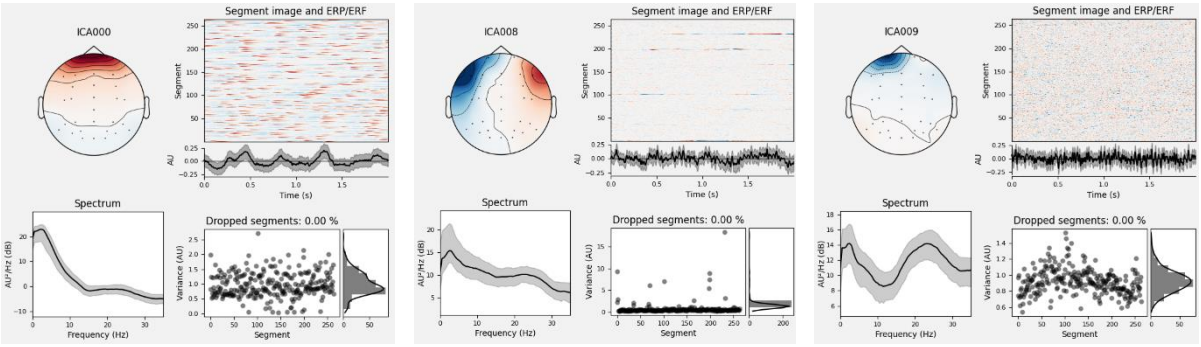


Figure 14: Properties of ICA components 000, 008 and 009 of subject 003.

The results of removing artifactual components of the ICA are checked with an ICA overlay plot. This epoched view is shown in figure 16. The original signal before removing ICA artifacts is drawn in red and the result is overlayed in black. It can be clearly seen that the big outliers are not anymore part of the signal after removing artifacts. Hence, it can be concluded that the artifact removal was successful.

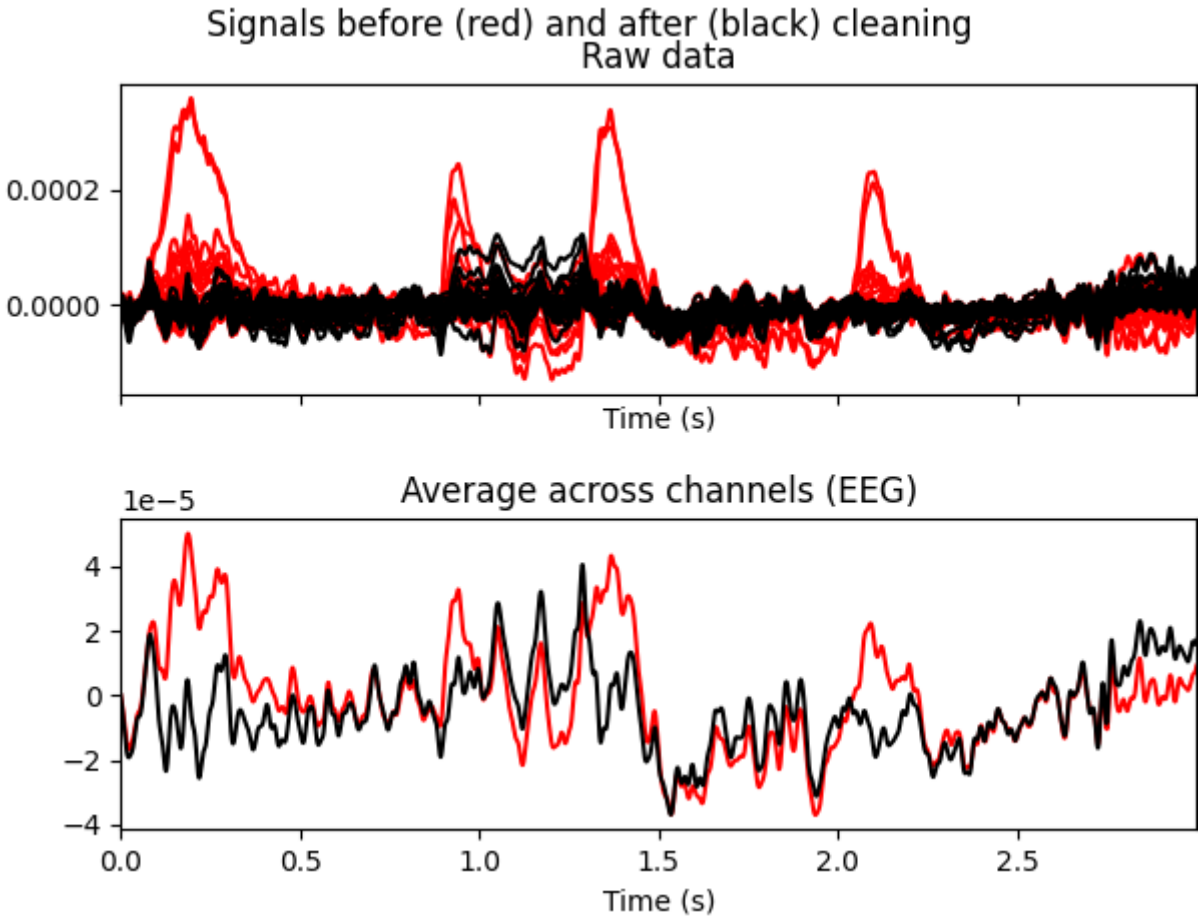


Figure 15: ICA overlay plot of subject 003, signal before cleaning (red) and after cleaning (black) is overlayed.

3.7.2.3 Subject 004

An overview of the ICA components of subject 004 is shown in figure 17. Figure 18 shows the property plots for ICA components 000, 003 and 009. All are considered as artefacts for the following reasons:

- 000: General eye artefact: ECDs near eyes, power concentrated at low frequencies
- 003: General eye artefact: ECDs near eyes, power concentrated at low frequencies
- 009: Not a clear eye artefact but power spectrum is strong at low frequencies

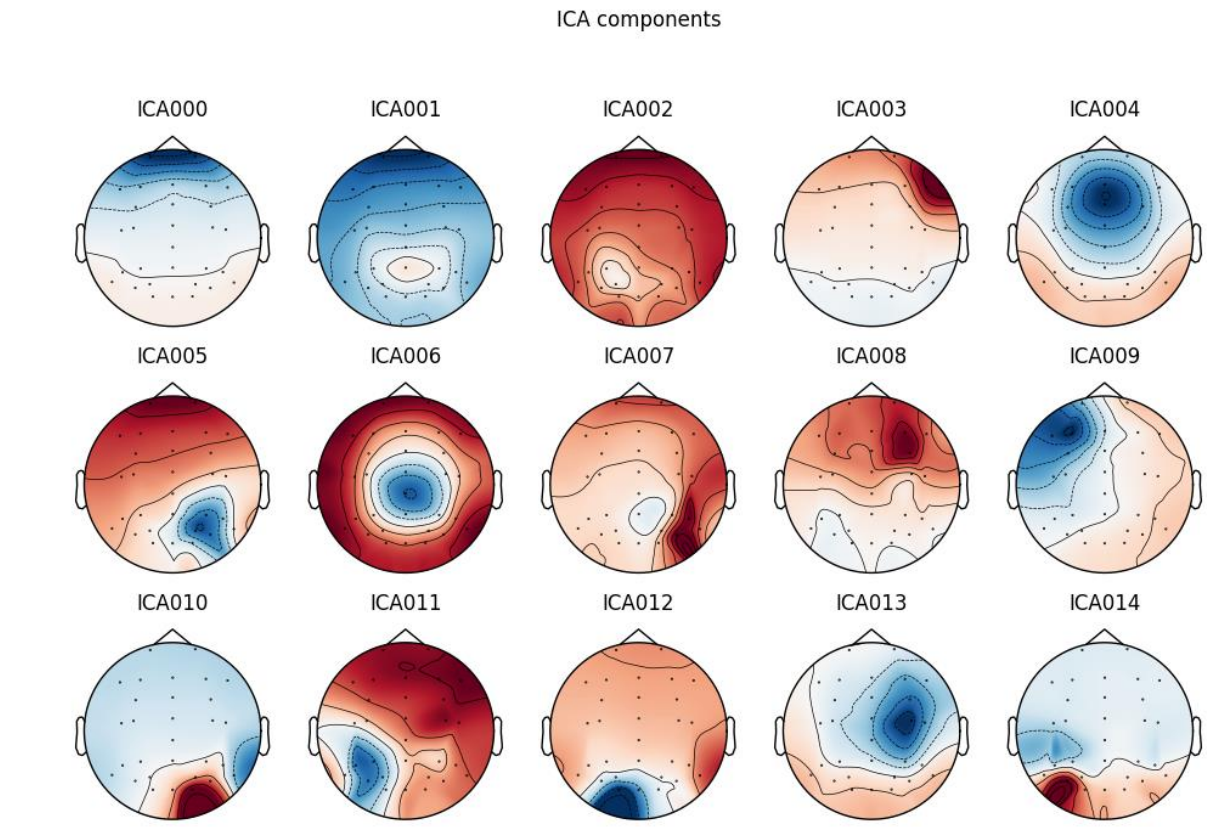


Figure 16: ICA components of subject 004.

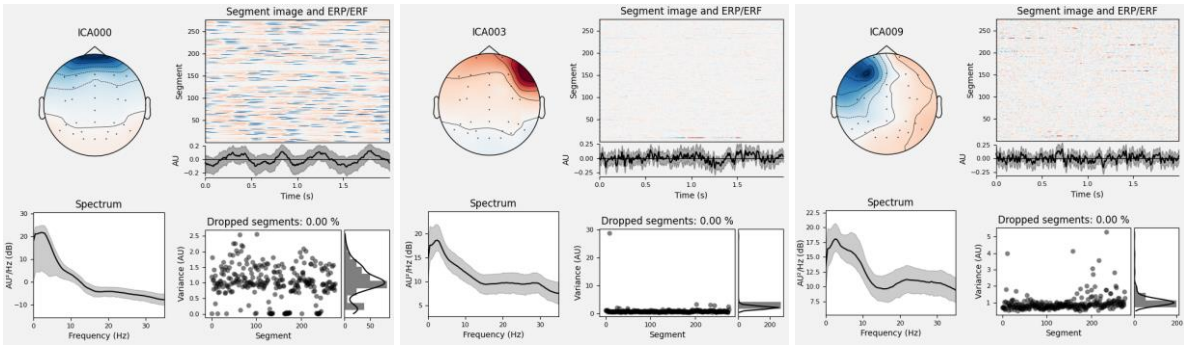


Figure 17: Properties of ICA components 000, 003 and 009 of subject 004.

The results of removing artifactual components of the ICA are checked with an ICA overlay plot. This epoched view is shown in figure 19. The original signal before removing ICA artifacts is drawn in red and the result is overlayed in black. It can be clearly seen that the big outliers are not anymore part of the signal after removing artifacts. In this case, probably not all incorrect outliers could be removed, since the start of the signal still looks pretty wide spread. It has been figured out at the very end of the project that the ICA is non-deterministic. Hence, for each run the order of components might be changed if the relevance of neighboring components is very even. Probably, in this case one of the three identified artifacts ended up with a different index and therefore was not removed. Setting the random seed will be considered in future work and can be easily done with:

```
mne.preprocessing.ICA(random_state=random_seed)
```

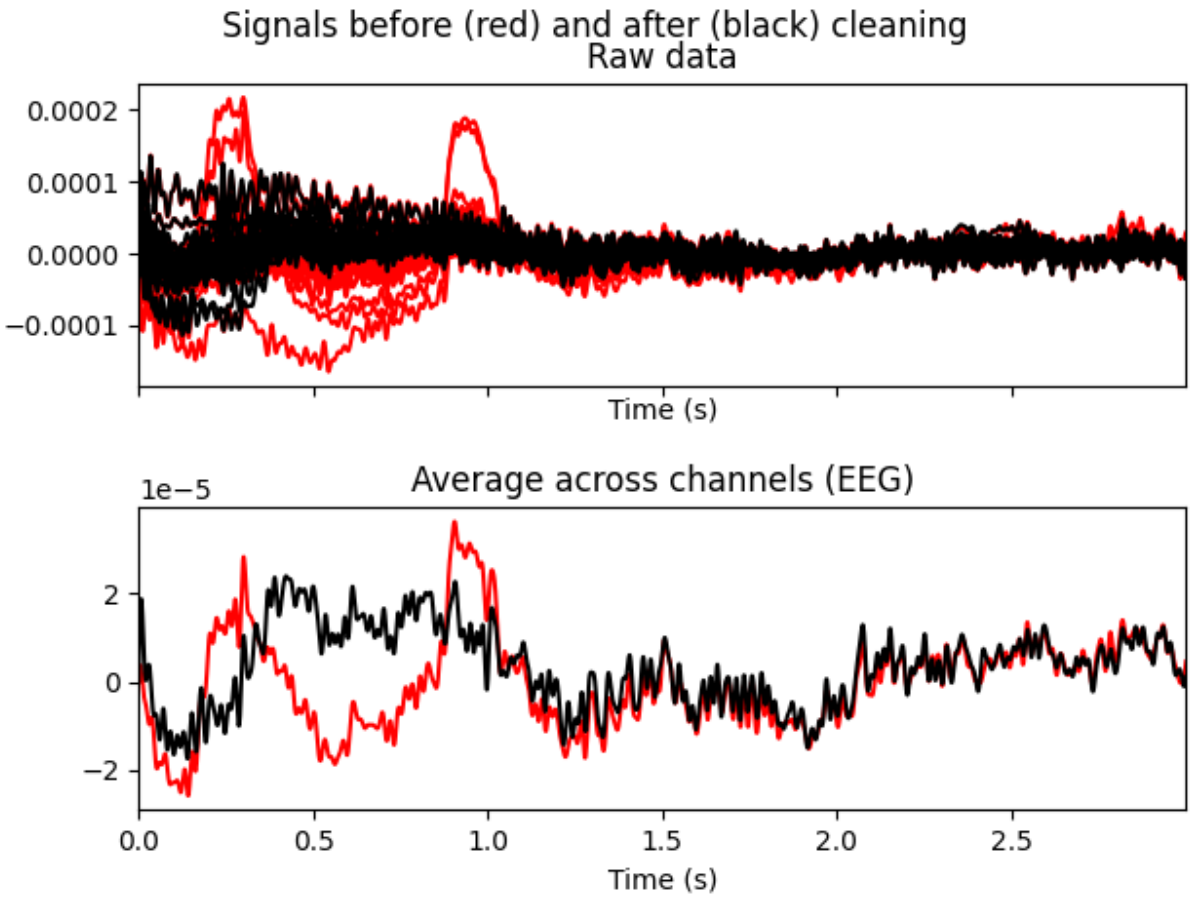


Figure 18: ICA overlay plot of subject 004, signal before cleaning (red) and after cleaning (black) is overlayed.

3.7.3 Apply ICA to exclude artifacts

The following MNE function takes care of applying the ICA and excluding the list of bad components in this process:

```
ica.apply(self.raw, exclude=bad_comps)
```

For the subjects 002, 003 and 004 the bad components are identified manually. For all other subjects the pre-computed bad components are used. It has to be considered that the pre-computed ICA components are identified based on a differently preprocessed signal. However, the operations done in preprocessing (e.g., filtering), are invariant with regards to ordering of the ICA components as stated in [6]. Hence, it is assumed to be safe to exclude bad components that have been enumerated from an ICA based on differently preprocessed data.

As for every step in the pipeline, also the effect of excluding ICA artifacts is sanity-checked. This has been shown for the manually cleaned subjects but was also performed for the pre-computed data. The MNE toolbox provides the following function to overlay raw and cleaned signals:

```
ica.plot_overlay(raw)
```

As a result, two plots are generated as shown in figure 20 for subject 040. The upper plot shows the raw data, the lower plot shows the average across all channels. The original data is drawn in red, the cleaning result is drawn in black. This overlay plot is helpful to visualize the effect of artefact rejection and the signal quality in general. It can be seen well in the overlay view that strong outliers are not visible after cleaning. The reason is that ICA components are rejected which represent undesired artefacts. These additive components are not part of the desired signal and must be subtracted, resulting in removal of outliers in the original signal.

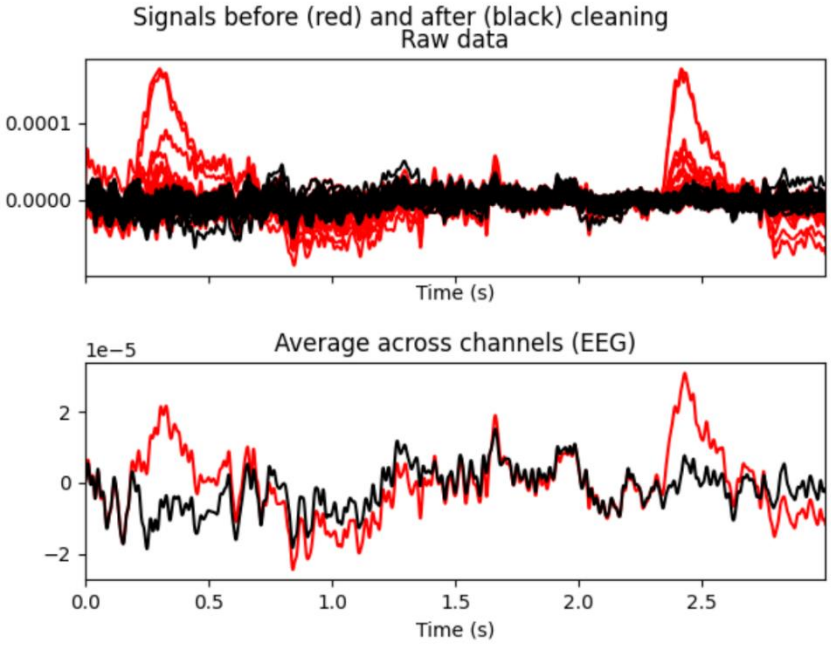


Figure 19: Overlay of raw and cleaned signals in ICA. Shown is subject 040.

3.8 Interpolate channels

Files: _01_clean_channels.py

As described in section “Bad channels”, noisy or dead channels are manually selected and marked. The annotations are stored in the info dictionary of the raw object:

```
raw.info['bads']
```

After identifying bad channels, they are interpolated as described in [5] with the following method:

```
raw.interpolate_bads()
```

As with every step in the pipeline, it is important to sanity-check this implementation. Hence, as a demonstration, the following channels were temporarily marked as bad for subject 001:

['F3', 'F7', 'FC3', 'C3', 'C5', 'P3']

The result can be found in figure 21. The upper plot shows all channels, including the channels with bad annotation in red. After interpolation, the originally red curves change and they are drawn in

black, since the list of bad channels is cleared. It is furthermore worthwhile to note that the channels of subject 001 are overall very noisy. Details can be found in the subsection “Bad subjects”.

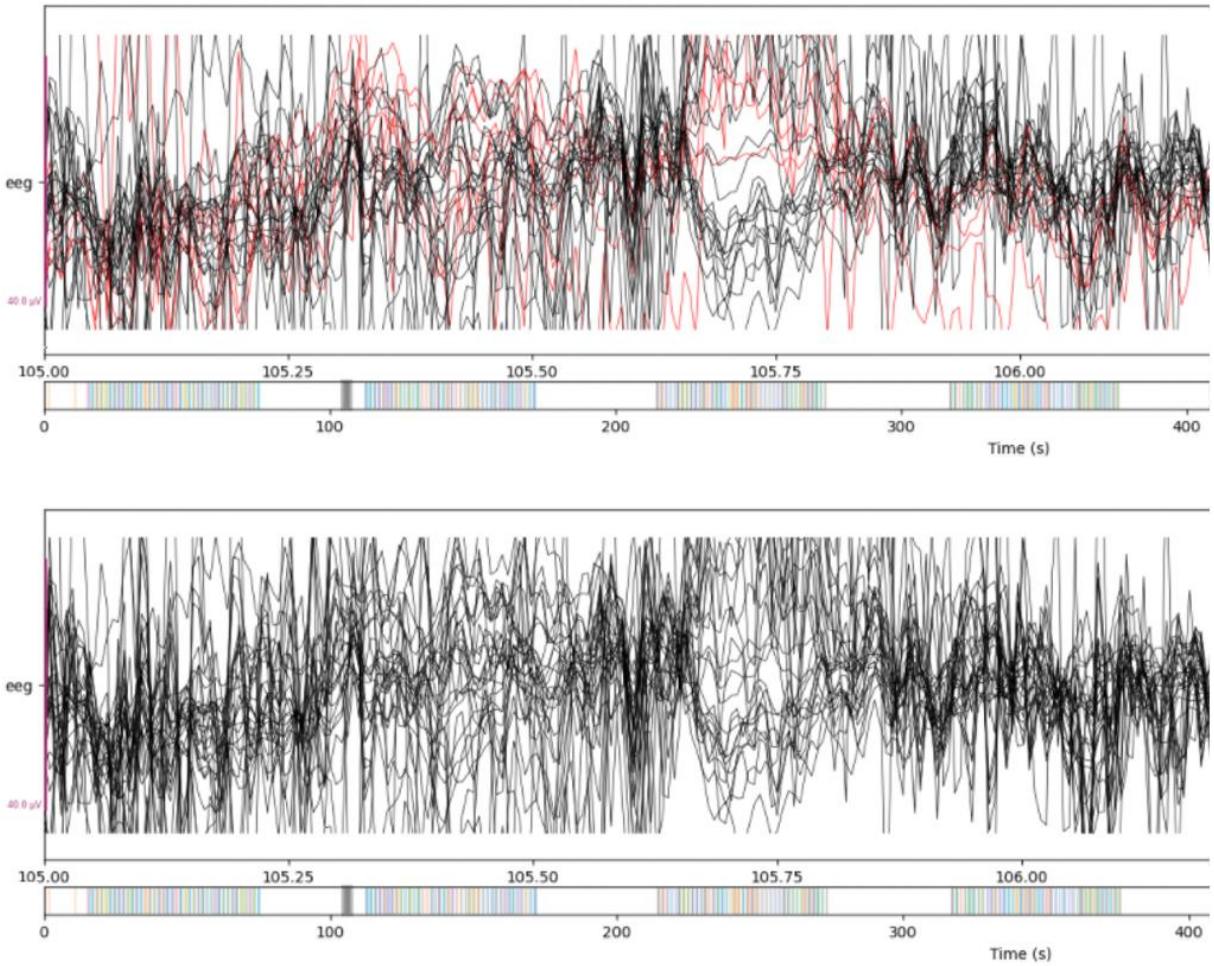


Figure 20: Butterfly view of selected channels of subject 001 before interpolation (upper) and after interpolation (lower). For demonstration purposes, the channels ['F3', 'F7', 'FC3', 'C3', 'C5', 'P3'] have been marked as bad.

3.9 Rereference

Files: _04_reference.py

Referencing is applied using the standard MNE function and using the average over all channels as a reference:

```
mne.set_eeg_reference(raw, ref_channels='average')
```

This re-referencing procedure is similar to what is applied in the ERP core paper: "*For analysis of the N170, the EEG signals were referenced to the average of all 33 sites (because the average reference is standard in the N170 literature)*." [1]

The effect is visualized for each subject in the report. Figure 22 shows the effect for subject 40 based on three channels. It is important to apply the same scaling to both plots in order to ensure comparability. In this work, the plot is generated with the following scalings:

```
scalings=40e-6
```

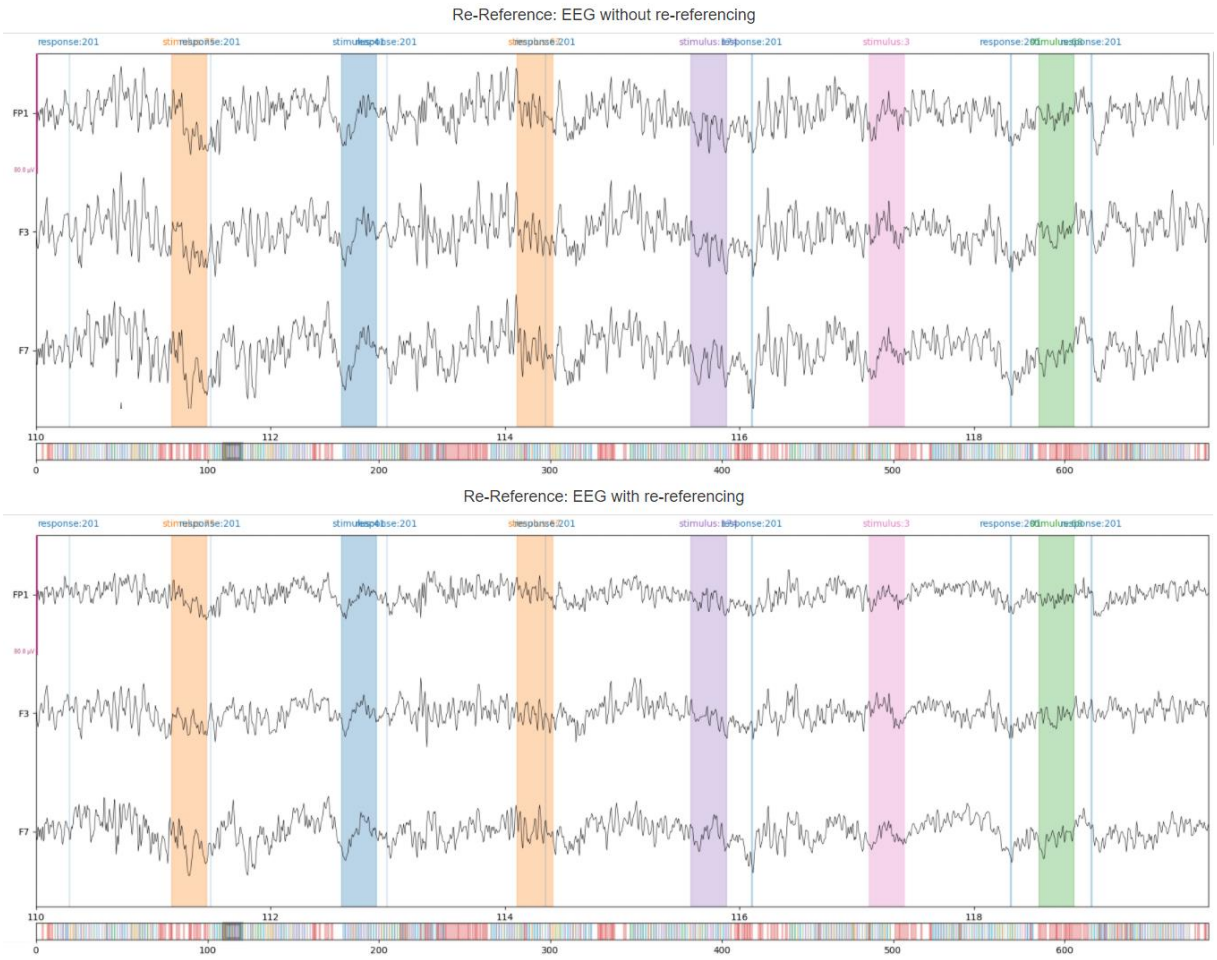


Figure 21: EEG without (upper) and with (lower) average re-referencing applied. Subject 040 is shown.

4 Analysis

4.1 ERP Peak Analysis

Files: _10_erp_peak_extraction.py, _20_erp_peak_analysis.py

This exercise requires to extract the study-relevant ERP peak subjectwise (e.g. one value per subject) and statistically test them. The underlying research question is: On which ERP-peaks do we find major difference between the conditions?

The peak analysis is performed on electrode PO8 and in a time frame between 130ms and 200ms after the stimulus onset. This matches the results of the work of Rossion regarding the N170 effect for faces [24].

```
erppeakanalysis:
  electrode: "PO8"
  crop_tmin: 0.13  # s
  crop_tmax: 0.2  # s
```

4.1.1 Difference wave and grand average

The difference wave is already studied per subject. Each subject report contains two plots. The joint plot over all channels (see figure 23) shows the epoched activity per channel. To ensure comparability across different subject reports, the time windows for the topoplot have been set to fixed values in the config.yaml:

```
difference_wave_times: [0.17, 0.21, 0.30, 0.41]
```

The times have been chosen based on expected activity in parent waves and difference wave.

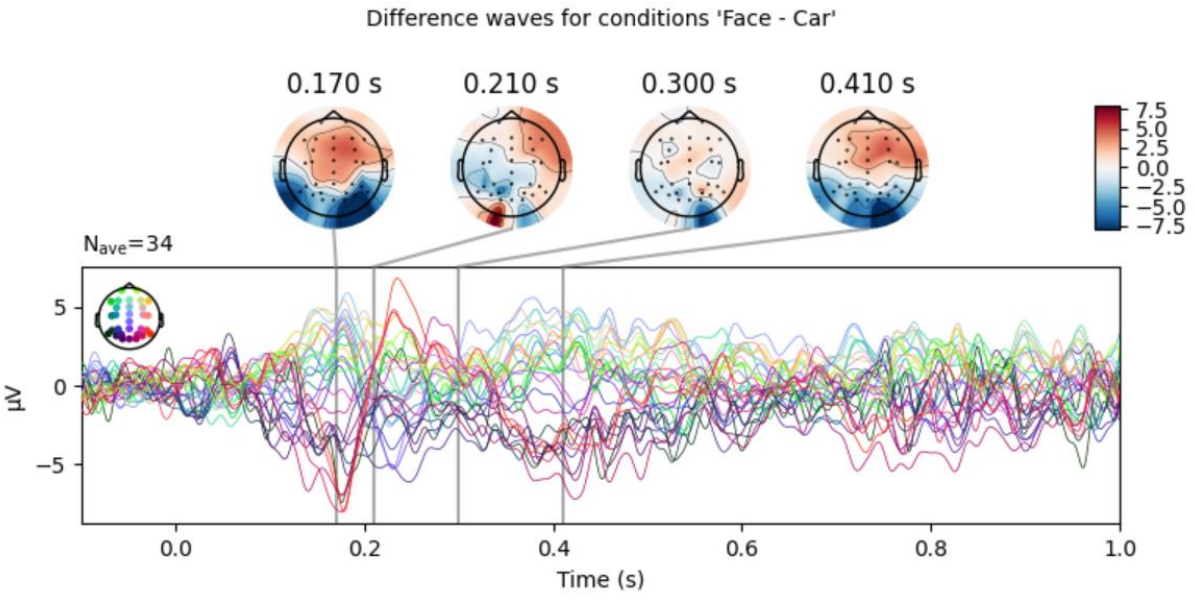


Figure 22: Joint plot over all channels for subject 040.

The second plot, that is carried out per subject, is the wave form per condition “faces” and “cars” as well as the difference wave for electrode PO8, where activity is expected [24]. An example can be seen in figure 24 for subject 040.

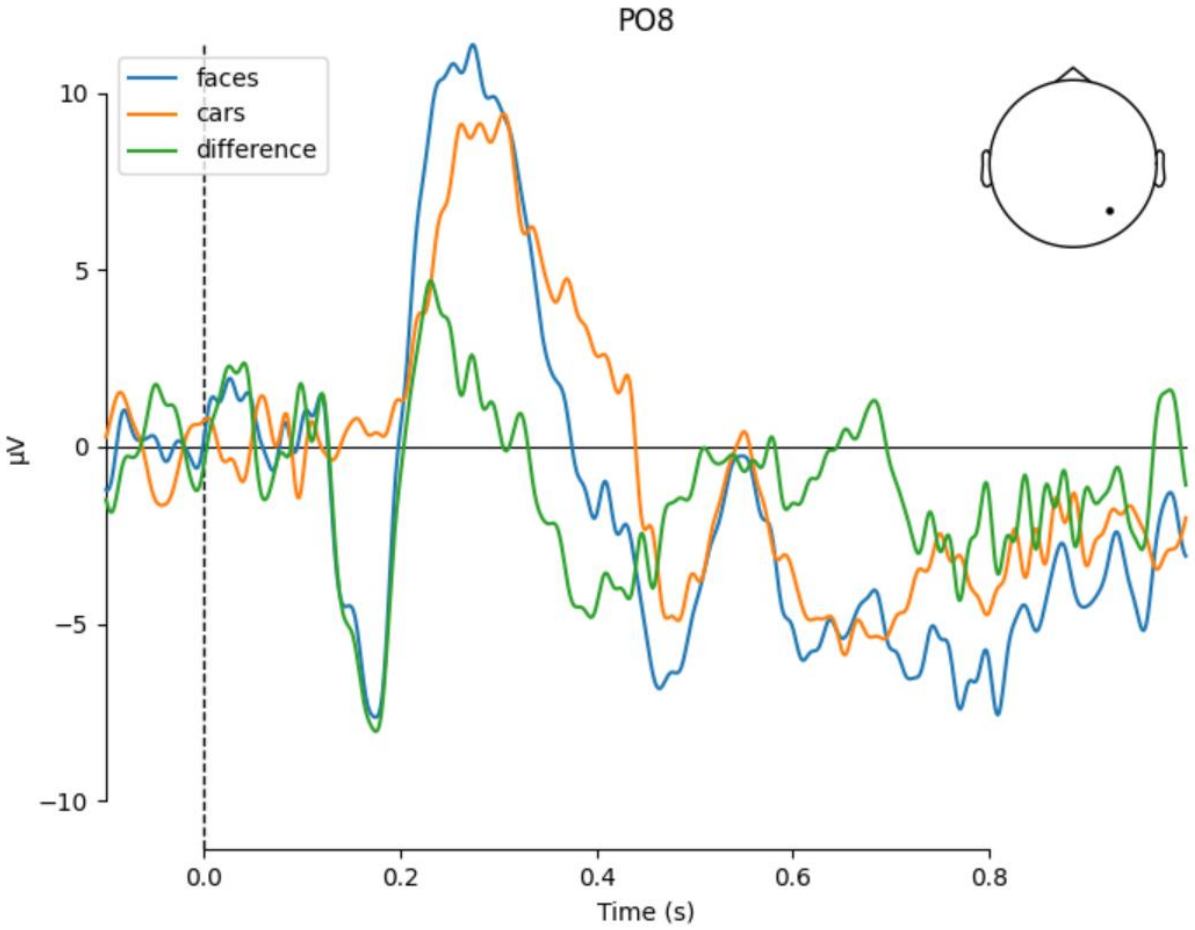


Figure 23: ERP waveforms per condition and difference wave for electrode PO8 of subject 040.

Figure 25 shows a comparison view over face and car stimulus of the activity across epochs. Even though this is only one subject, hints can be found that the N170 effect has slightly stronger amplitude and starts slightly earlier for faces than for cars.

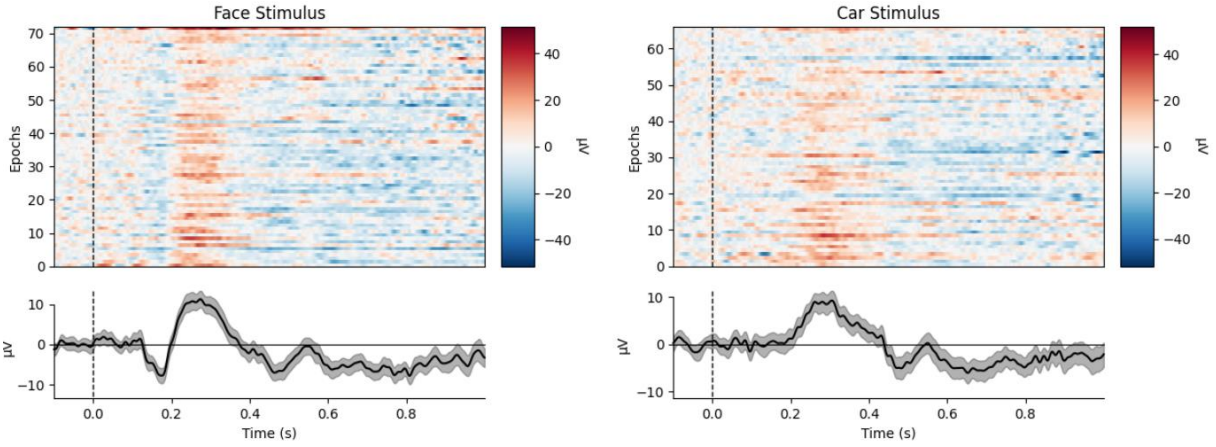


Figure 24: Activity over epochs per condition for subject 040.

The signal at electrode PO8 per condition is then averaged across all subjects to get a grand average. Due to the good quality of data, that has been inspected on subject level, no robust statistics was necessary to achieve the expected shape of the curve. The resulting waves of grand average (across all subjects) for the conditions “faces” and “cars” is shown in figure 25. The difference wave is added in green. The qualitative and quantitative shape of the curves match the results of the ERP core paper [1] pretty well. As a comparison, their result is shown in figure 26.

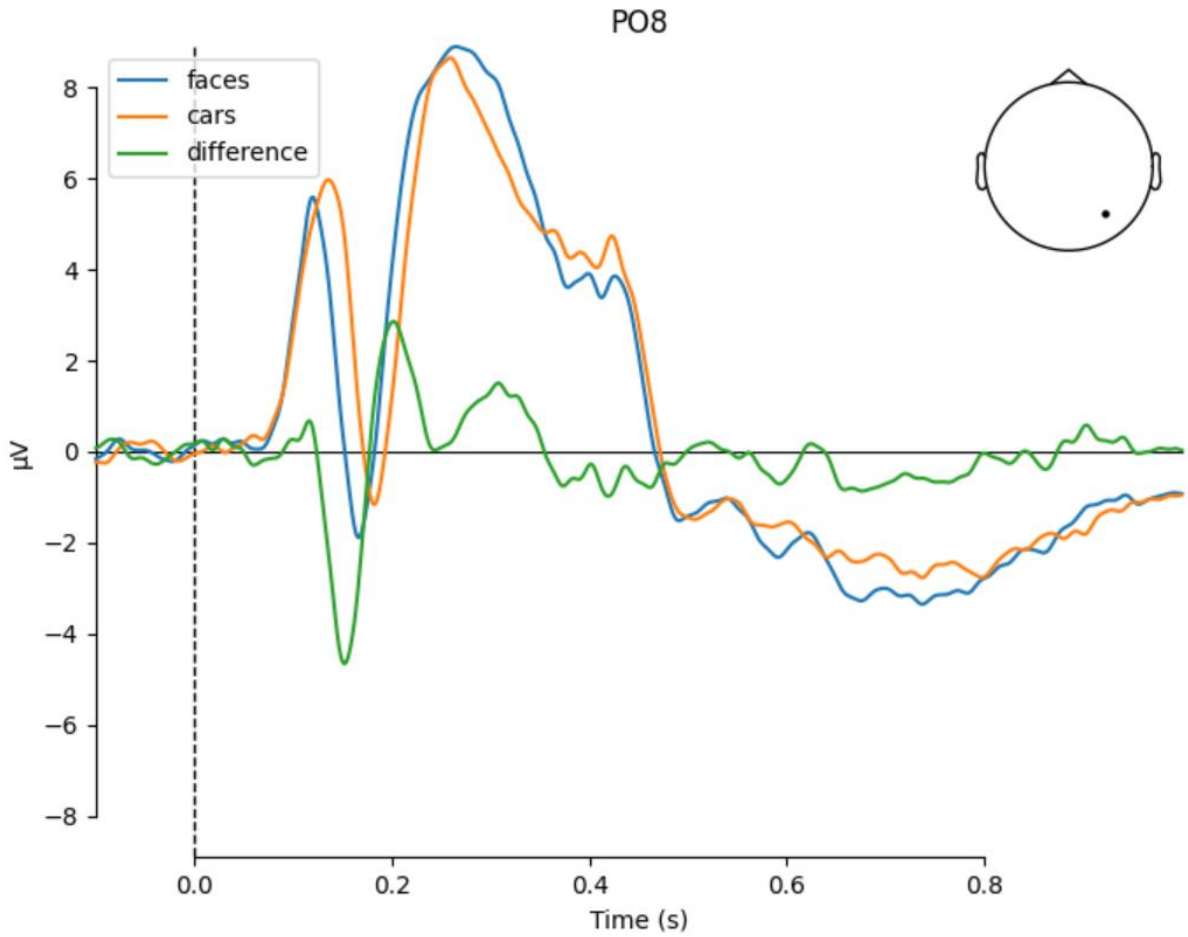


Figure 25: Grand average across all subjects for conditions and difference wave.

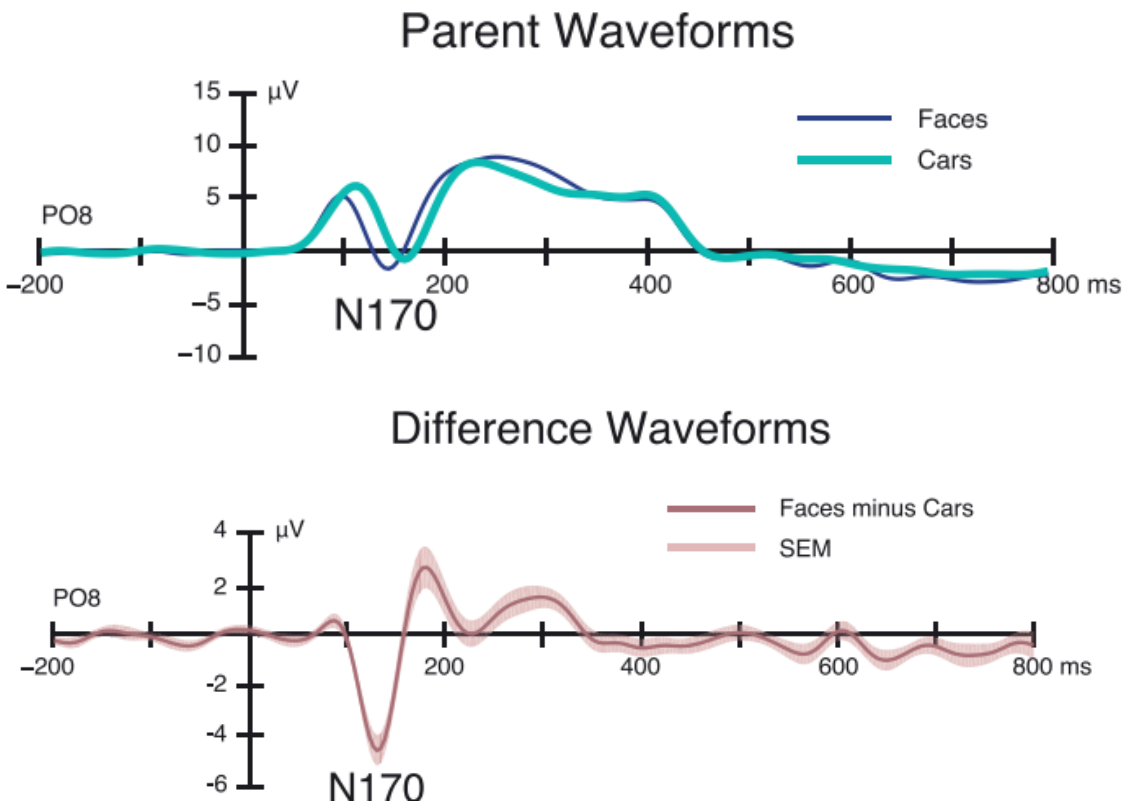


Figure 26: Extracted ERP waveforms for the N170 dataset. Found in the ERP Core paper [1].

Valuable insights are furthermore given by looking at the distribution of the activity over the sculp. Hence, a topo map of the average difference wave is generated as shown in figure 27. It can be seen that there is few activity before 150ms. Then, the peak occurs as a contrast in potential between the center and the back of the brain. The polarization inverts at 200ms and then levels off. Having this topo map in conjunction with the difference wave is very helpful to locate the source of the activity. It confirms the proposal of Rossion to evaluate electrode PO8 for the N170 effect [24].

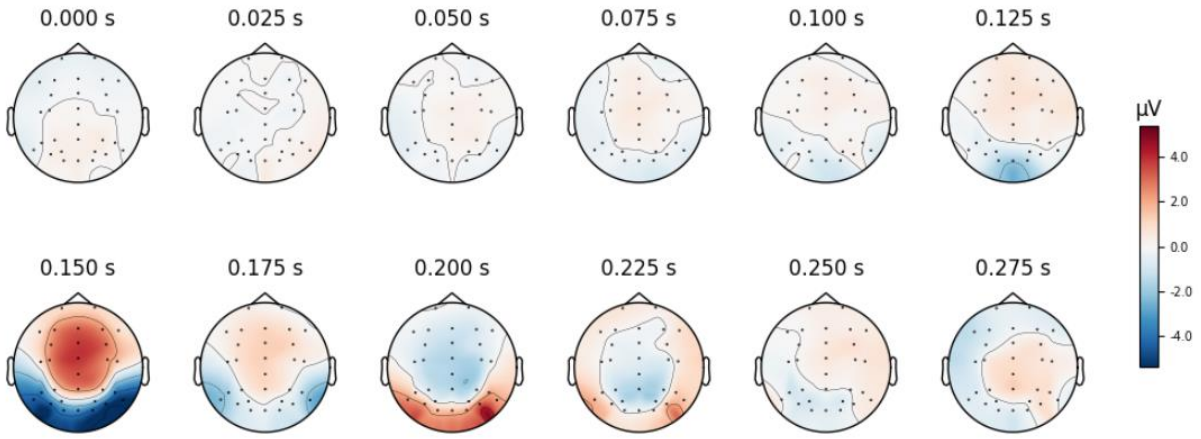


Figure 27: Topo map of average difference wave at different times.

4.1.2 Trials with wrong response

An important question in this analysis was how to deal with wrong responses. When a subject didn't give the correct response in one trial, this epoch needs to be treated with caution. The subject for example could have been distracted at this point in time. It is also possible that the subject pressed without really processing the task. In both cases it is not ensured that the subject actually had the desired thought process that would trigger the brain activity under study. Hence, it was decided to remove trials with wrong response. This was generically done for all steps in:

```
Base.load()
```

As a sanity check, the change was evaluated on the joint epoch plot, see figure 28. The upper plot shows the epoched joined activity for all channels before wrong responses have been cleared. The lower plot shows the same after clearing wrong responses from the epochs. From a qualitative point of view the overall activity looks much clearer after clearing wrong responses. The assumed reason is that a wrong response implies distraction or mental absence. In both cases, the induced brain activity does not include the effect under study. Hence, the trials with wrong response are removed.

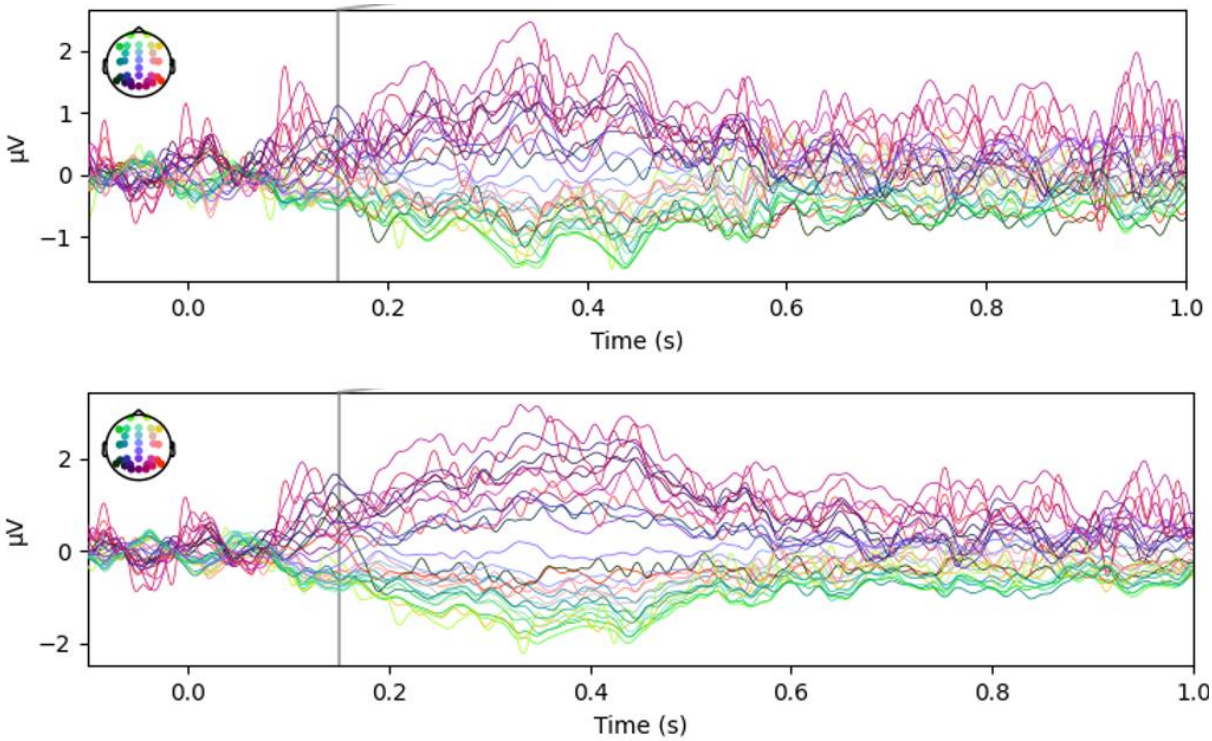


Figure 28: Joint epoch plot before removing trials with wrong response (upper) and after (lower).

4.1.3 Robust statistics

During sanity-checking the ERP peak data, a plot of the distribution over time was created. It could be figured out that the underlying distribution must be bi-modal with an expected peak at around 150ms-170ms and an additional peak directly at the edge of the chosen cropping time of 200ms. When plotting the time values together with their corresponding difference of event potential (see figure 29), the picture was much clearer. The data can be clustered into 2 clusters. Circled in orange are the data points that mainly result from the N170 activity with negative electric potential. In the

positive region, however, another cluster is formed (marked in green). Those values are positive, indicating a positive peak. All data points are at the high side with regards to the offset time. Hence, it is assumed, that those data points stem not from the N170 peak but from the subsequent positive peak.

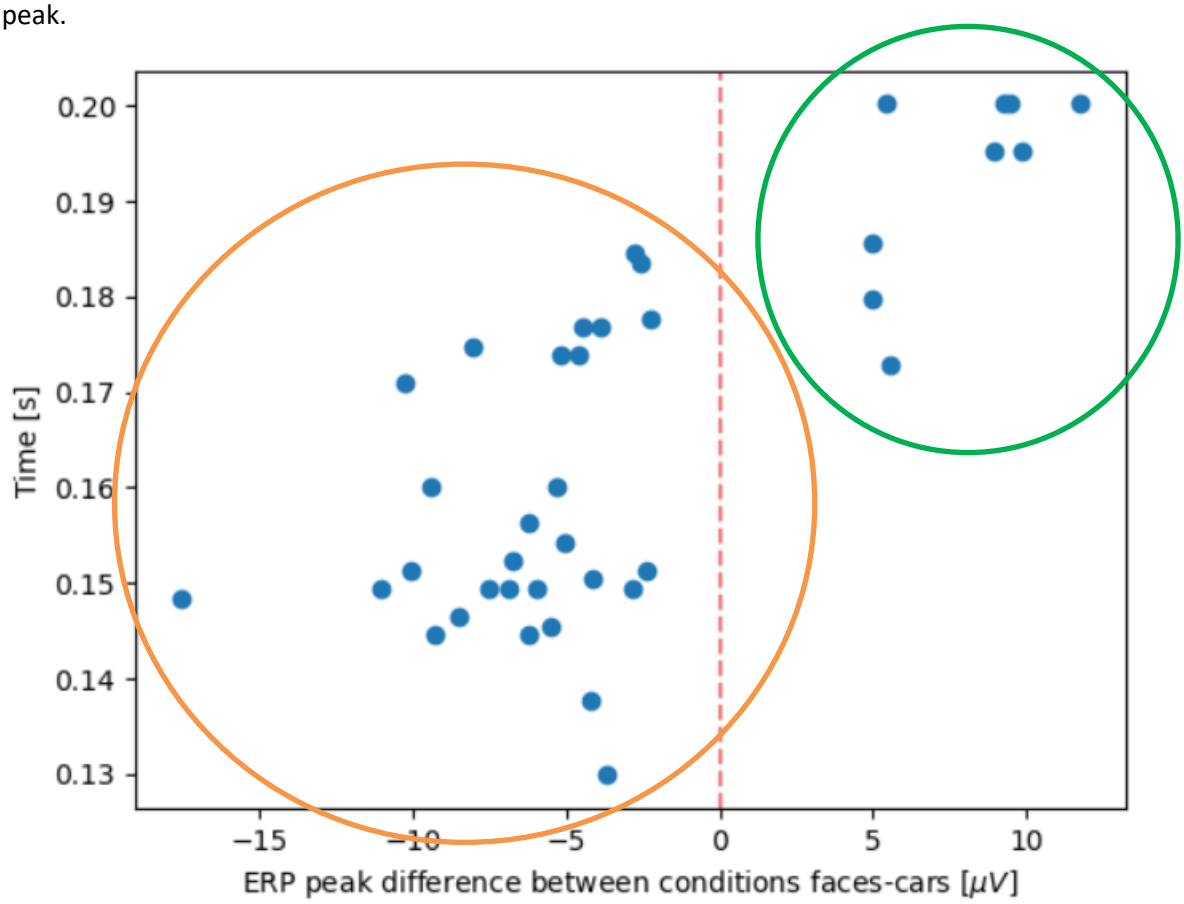


Figure 29: Scatter plot to visualize correlation between ERP peak difference and time.

A main reason for this is that the measured time of events does not match to the setting for the subject. The reason has been identified by the researchers of the ERP core paper to be the delay of the LCD monitor [1]. In the Readme file for the N170 dataset the delay is quantified with 26ms [18]. When cropping between 130ms and 200ms without accounting for the 26ms shift, in some instances the following positive peak is extracted. An example is given in figure 30, where the difference wave is plotted for subject 027. It can be seen that the N170 peak is pretty small with $\sim -2\mu V$. The subsequent peak in positive direction reaches $\sim 5\mu V$. Due to the missing time shift, this peak is still in the cropped time range of the epoch (visualized in purple). This is a representative example of cases, where the 26ms delay perfectly explains the outlier peaks, because in those instances the subsequent peak in positive direction falls into the cropped time frame.

Based on this knowledge, one could apply a shift and easily remove all problematic data points. However, this has a characteristic of tweaking data, which also introduces potential side-effects to the statistics (e.g., fishing for significance). For this reason, in this work the concept of robust statistics is applied to handle these outliers. The data series is winsorized with a limit of 25% for the both the upper and lower end of the ordered values. Values that are out of the range are set to the 25%-percentil (lower) and the 75%-percentil (upper) accordingly. The outcome of this robust statistics is discussed in the subsequent section “Significance testing”. Using a windsorized limit of 25% corresponds to a trimmed mean of 50%. This leaves still much data in and serves as a good

balance between the mean (0% trimmed mean) and the median (100% trimmed mean). Because it applies: “Fun fact: a 100% trimmed mean is just the median!” [20].

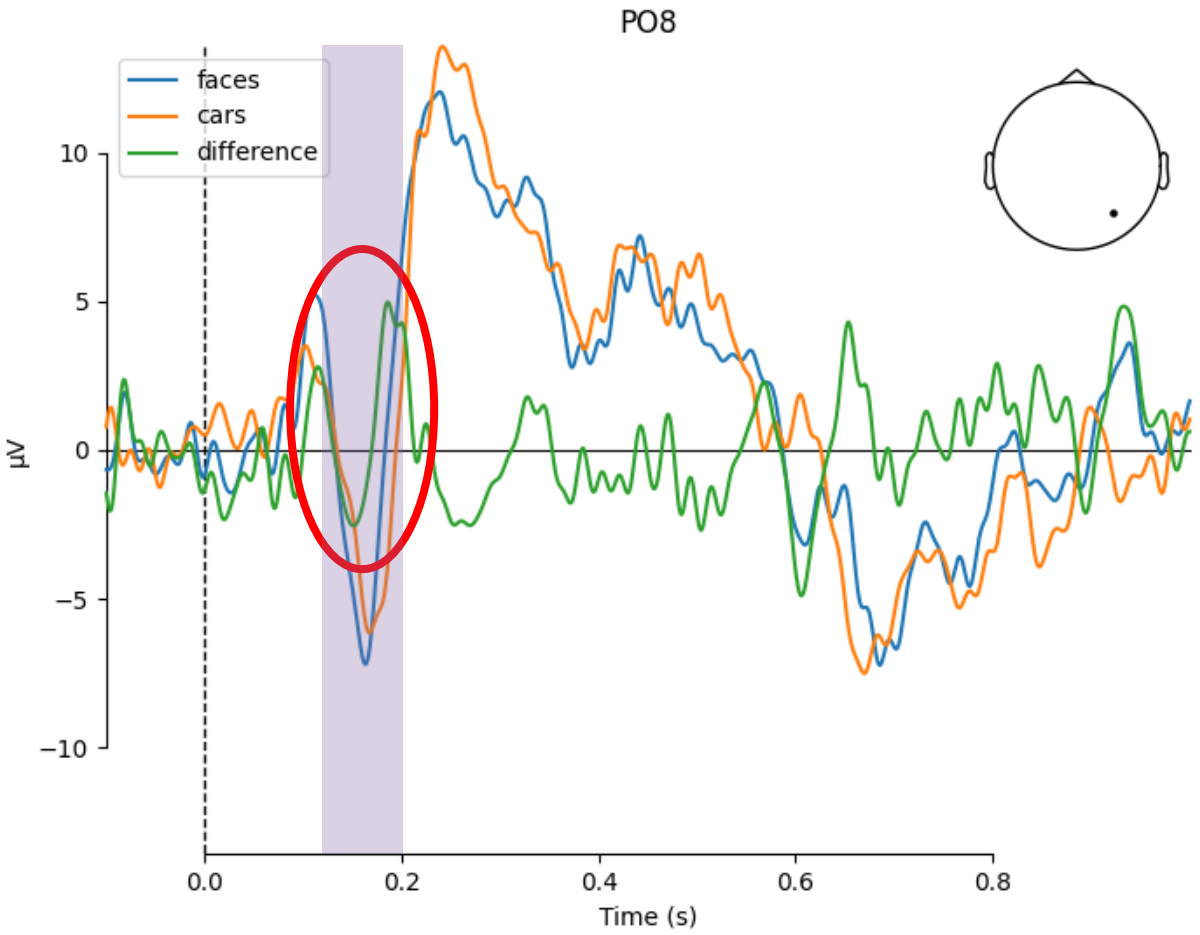


Figure 30: Comparison of evoked potentials for conditions faces and cars, Subject 27.

4.1.4 Significance testing

To test the significance of the difference between conditions “faces” and “cars”, the peaks of the differences are extracted in the cropped window between 130ms and 200ms as proposed by Rossion [24]. The following null and alternative hypotheses are constituted for testing:

H_0: $peak_{difference\_wave} = 0.0$ i.e., the peak of the difference wave has amplitude 0, no information about the condition is available in the difference wave.

H_1: $peak_{difference\_wave} != 0.0$ i.e., the peak of the difference wave has amplitude different from 0, information about the condition is available in the difference wave.

A conventional two-sided one sample t-test is used to test if the mean of our peaks of the difference wave is different from the hypothesized value 0.

The result is shown in figure 31, where a histogram of the peaks of the difference wave between the conditions “faces” and “cars” is given. On the left side, the actual data is used without robust statistics. As already illustrated in the chapter “Robust statistics”, two modes are visible. They are produced because for some subjects the subsequent, positive peak is found. The reason is a missing application of LCD monitor delay, as discussed before. Still, the t-test is significant with a p-value of

0.011 and a significance level of 0.05. The mean of the peak is calculated as $t_{peak} = 0.166s$, which matches the expectation, that the difference in condition is related to the N170 effect for faces. The mean however has a slightly smaller amplitude than expected with $-2.96\mu V$.

Applying robust statistics, the values change. The distribution is sharpened around the median and the outliers with positive electric peaks are substituted by the 75% percentil, which is already negative. The winsorized mean of the peak times remains roughly equal with $t_{peak} = 0.164s$. However, the mean of electric peak is now $-4.56\mu V$, giving now a much more precise image of the actual peak of the difference wave. The reason can be inferred from the scatter plot in figure 29. Imagining the marginal distributions, it is clear that the distribution over electric potentials is bi-modal while the distribution over time is closer to a normal distribution. Hence, after applying robust statistics, the mean time peak is roughly equal but the mean of electric potential changes strongly. This also affects the t-test, resulting in a p-value almost at 0.0.

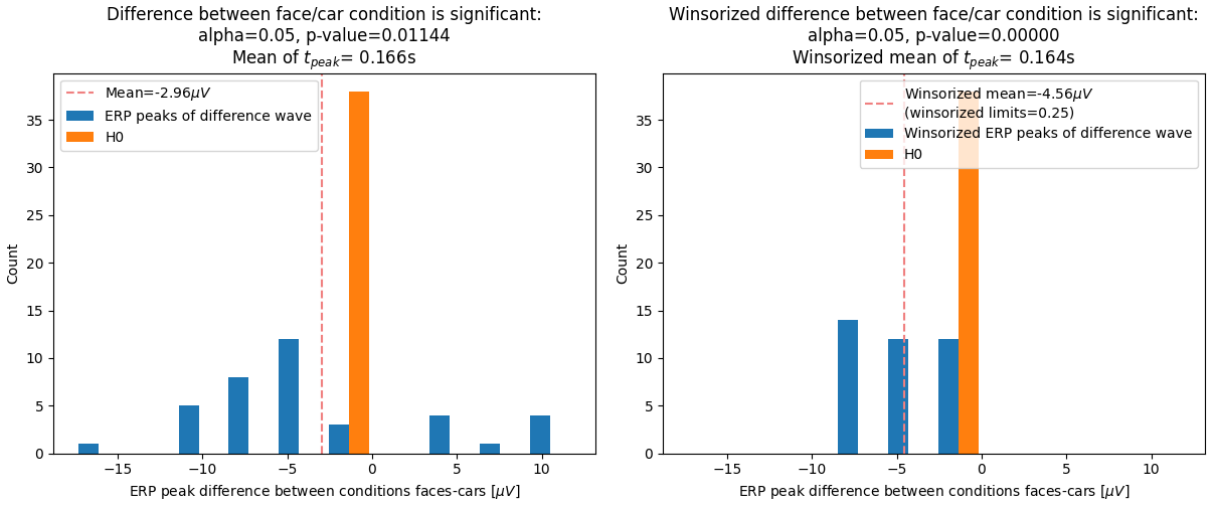


Figure 31: Histogram of ERP peaks of the difference wave, normal (left) and winsorized (right).

4.1.5 Summary

In the previous section it could be shown that the provided N170 dataset has a measurable effect at $t_{peak} = 0.164s$. This effect represents the difference wave between the conditions “faces” and “cars”. The winsorized mean of the difference peak is $-4.56\mu V$, using 25% as limits. This matches well with the values given in literature [1], as shown in the plot of the difference wave in figure 26, as well as with the grand average curve computed in this work (figure 25).

A one-sided one sample t-test with significance level of 0.05 indicated the significance of the effect with a small p-value of 0.011. Using winsorized data, this p-value turned out to be even smaller.

4.2 Decoding Analysis

Files: _11_decoding _extraction.py, _21_decoding_analysis.py

In the decoding analysis the goal is to decode the main contrast of the experiment across time. The following research question shall be discussed: When is information about the conditions in our data available? In order to answer this question, the main contrast of the experiment is decoded across time and tested for significance.

4.2.1 Decoding across time

Initially, a transformation of the full time series into CSP feature space has been evaluated. A result for electrode PO8 is shown in figure 32. The condition “faces” is marked in green, “cars” is red. It can be easily seen that the CSP is not able to find a transformation that clearly linearly separates the conditions. Two reasons could be identified: First, the CSP feature transform is only useful for oscillatory activity. This is not necessarily the case for the ERP activity under study. Also, when looking at the grand average wave, the feature of interest is not easily extractable from just the overall data, since the behavior of the curves of both conditions are very similar and only become obvious when subtracting the conditions, i.e., looking at the difference wave. Hence, a suitable feature extractor should be embedded into a sliding window approach.

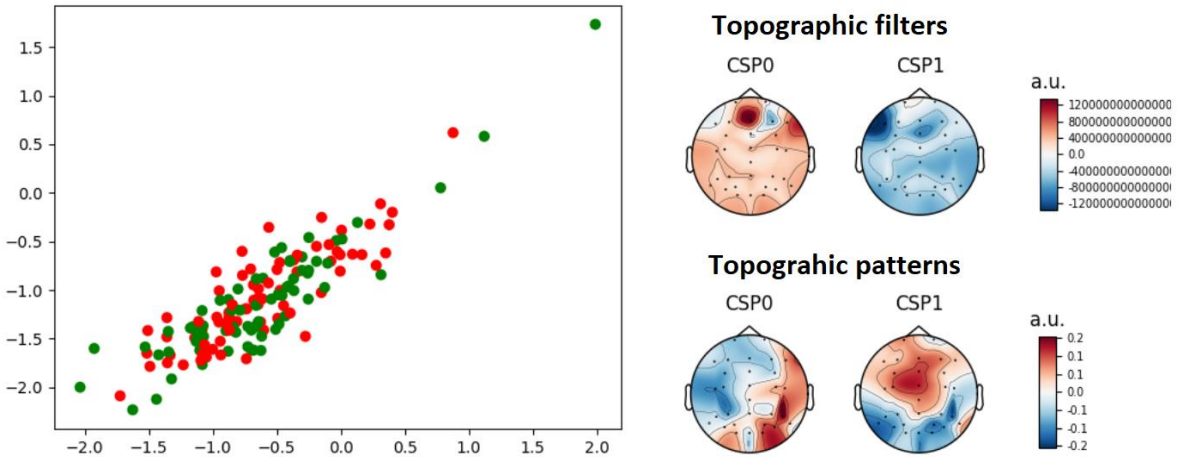


Figure 32: 2-dimensional feature space of CSP (left) and topographic filter and pattern (right) without decoding over time. Transformed was electrode PO8 from subject 003.

In the next step, the feature extractor is embedded into a sliding estimator as follows:

```
mne.decoding.SlidingEstimator(pipeline, scoring='roc_auc')
```

The score used is the area under curve (AUC) of the receiver operating characteristic (ROC) curve. This score is commonly used for optimization in binary classification tasks. The result of this sliding estimator is a score value per timestep. This can be rather noisy on a subject level. An example is given in figure 33 for subject 39. Qualitatively, most of the subjects have a peak in score at around

170ms, indicating that most information entropy is available at that point in time. Also, the score fluctuates at around 0.5 (chance) before 100ms-150ms.

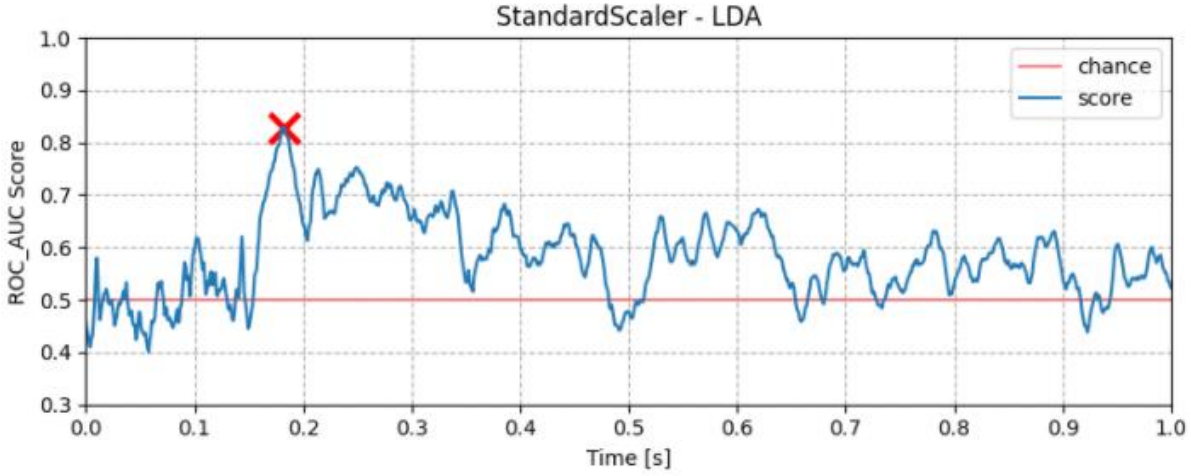


Figure 33: Decoding score across time for Subject 39.

4.2.2 Pipeline and cross-fold validation

To automate the training and testing procedure, an sklearn pipeline is used. The pipeline has two steps, a feature extractor and a classifier. Multiple combinations of feature extractors and classifiers are evaluated in following subsections.

One key thing to avoid when learning from data is overfitting, which means tuning the parameters to only perfectly score on the training data. Instead, one aims to train a model that generalizes and hence works well on unseen data [13]. In this work, a technique called k-fold cross validation is used:

```
sklearn.model_selection.StratifiedShuffleSplit()
```

The data is randomly shuffled and split 10 times. Each time, 20% of the data is held out as test data. The classifier is applied to the training data and scored on the test data. This way, the score reflects the generalization capability of the trained model. Finally, the score is averaged across each of the 10 folds, resulting in an overall test score.

```
mne.decoding.cross_val_multiscore().mean(axis=0)
```

4.2.3 Analysis of feature extractors and classifiers

Multiple scoring strategies are evaluated. For feature extractors, StandardScaler (equivalent to mne.decoding.Scaler(scalings='mean')) and Vectorizer have been considered. Unfortunately, CSP and SPoC did not work due to a bug with MNE, where the execution caused a PermissionError for a temp file that is created at execution. As classifier, LDA, Logistic Regression and Support Vector Machine (SVM) are used from the sklearn package. The combinations under evaluation are defined in the config.yaml as follows:

```
decodinganalysis:
  scoring_strategy:
    - "StandardScaler-LDA"
    - "StandardScaler-LogisticRegression"
```

```
    - "StandardScaler-SVM"
    - "Vectorizer-LDA"
    - "Vectorizer-LogisticRegression"
    - "Vectorizer-SVM"
```

The pipeline automatically calculates the scoring over time per strategy for each of the subjects. In the analysis step, this subject-specific data is aggregated per strategy and significance tested. A comparison of the different strategies can be found in the subsection “Result plots”.

4.2.4 Significance testing

The scores are provided from the previous steps per sampled timestep. To do significance testing, they are averaged with $t_{sampling} = \Delta t = 20ms$. This way the noise can be reduced by still maintaining a sufficient resolution in time. Averaging is considered to not be problematic since the decoding across time anyways works with a sliding window approach to include the near past. Smaller values for $t_{sampling}$ have been evaluated but not considered advantageous.

A statistical test is performed per timestep Δt to test for the following question: At which timesteps is statistically significant information available?

$\mathrm{H}_0(\Delta t)$: $score(\Delta t) \leq 0.5$ i.e., the score is smaller or equal than chance, no information about the condition is available at $t = \Delta t$.

$\mathrm{H}_1(\Delta t)$: $score(\Delta t) > 0.5$ i.e., the score is larger than chance, information about the condition is available at $t = \Delta t$.

A reduced alpha value of $\alpha = 0.025$ is used as a significance level for the test. The reason is that in binary classification tasks it can easily happen to find a model that by chance reaches a slightly better score than 50% on average across all subjects. Since we are evaluating multiple models in k-fold cross validation, the best-scoring model is used. It could be possible, that the model exploits some correlation in the data to get significantly better than chance. However, some correlation does not necessarily proof that there is in fact causal information that has been exploited. In addition to the weak punishment in binomial classification, the area under curve (AUC) of the receiver operating characteristic (ROC) curve is used. This score is good for optimization in binary classification tasks but does not punish false classifications as strongly as e.g. the F1 score. Due to the above reasons, a significance level of 0.025 is used.

A second consideration was to equalize event counts. The problem is that with unequal event counts, the classifier can learn the prior distribution of both classes. If the data is distributed very unequal, this can give very good scores in binary classification without actually learning the likelihood of the data, i.e., improving the prior belief by integrating evidence. Scoring based on the prior is obviously a confounding effect that does in reality not help to answer the research question: When is information about the conditions in our data available?

On the other side, equalizing event counts requires to drop events and hence equalize the relevance of all subjects relatively to each other. It is not desired to drop information on the one side and to consider a subject with very few trials equally important as a subject with many trials. This, however, is less relevant for pure decoding analysis compared to the described issues in the previous

paragraph. Having unequal event counts for binary classification does not allow to test significance by comparing to chance. The classifier can learn the uneven distribution as a prior and will score above 0.5 without having an actual effect at that specific time, or without seeing data at all. Hence, for this specific decoding analysis the event counts are equalized. For all other analyses in this work the event counts are not equalized.

Generally, this decision is not really critical in the N170 dataset, since the events are quite evenly distributed. Equalizing events between the conditions “faces” and “cars” result in dropping 9 epochs, which is a relatively small number. In this work, both alternatives have been evaluated and only a very minor change has been found.

The event counts are equalized using the MNE function:

```
epochs.equalize_event_counts(["faces", "cars"])
```

Detailed results per strategy can be found in the section “Results”. The general observation is as follows: With every strategy combination a significant effect has been observed given a significance value of 0.025. The effect was significant in the time range $0.1\text{s} < \Delta t < 0.9s$ consistently over all strategies. This is plausible, since no significance is reached before an actual brain activity can happen based on the observed image in the experiment (causality with regards to delay between eye and brain). Also, the significance ends at around 900ms, since most of the induced activity fades out there. The significance is visualized with a separate colored time axis, where red represents no significance and green represents significance.

4.2.5 Effect size

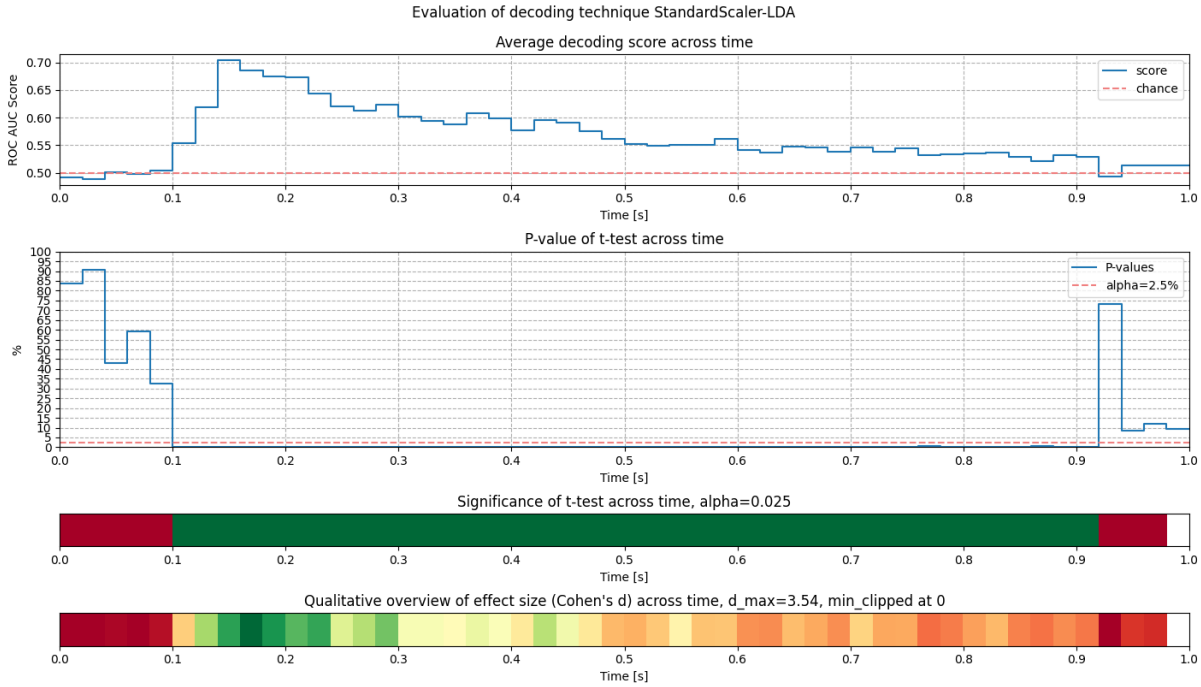
In the previous section it was investigated that a significant effect is available at certain time steps. This does not give any information about the size of the effect. Especially at later points in time after the onset event, the achieved score of the classifier might still be significant. On the other side, the average score might only be slightly above 50% (chance). This means that there might still be some significant information about the condition available at that time. On the other side, the effect is so small that it is barely relevant. To make this transparent, two plots were generated. First, the average decoding score across time is given. This score already tells, how much information is available at a specific point in time on average. Secondly, Cohen’s d was calculated as a measure of effect size and plotted right below the significance plot. It is plotted qualitatively, where red correspond to an effect size smaller or equal to 0 and dark green corresponds to the highest effect size of the time series. Additionally, the peak effect size is provided as a number in the title of the plot.

It can be seen very well that the effect size is roughly 0 if the time range is not significant. For the significant time, i.e., between 100 and 900ms, the effect size varies strongly. It rises up to a very strong peak in effect size at around 170ms, confirming the results of the ERP peak analysis before. Then it slightly decreases and reaches a second peak at around 220ms. This perfectly matches the second peak of the grand average difference wave generated in the ERP peak analysis. From this point in time, the score slowly decreases and reaches ~55% on average at 500ms. This score is only slightly above chance; hence the effect size is small, indicated with orange color. At around 900ms, the effect size gets close to 0, and also the significance level is not reached anymore.

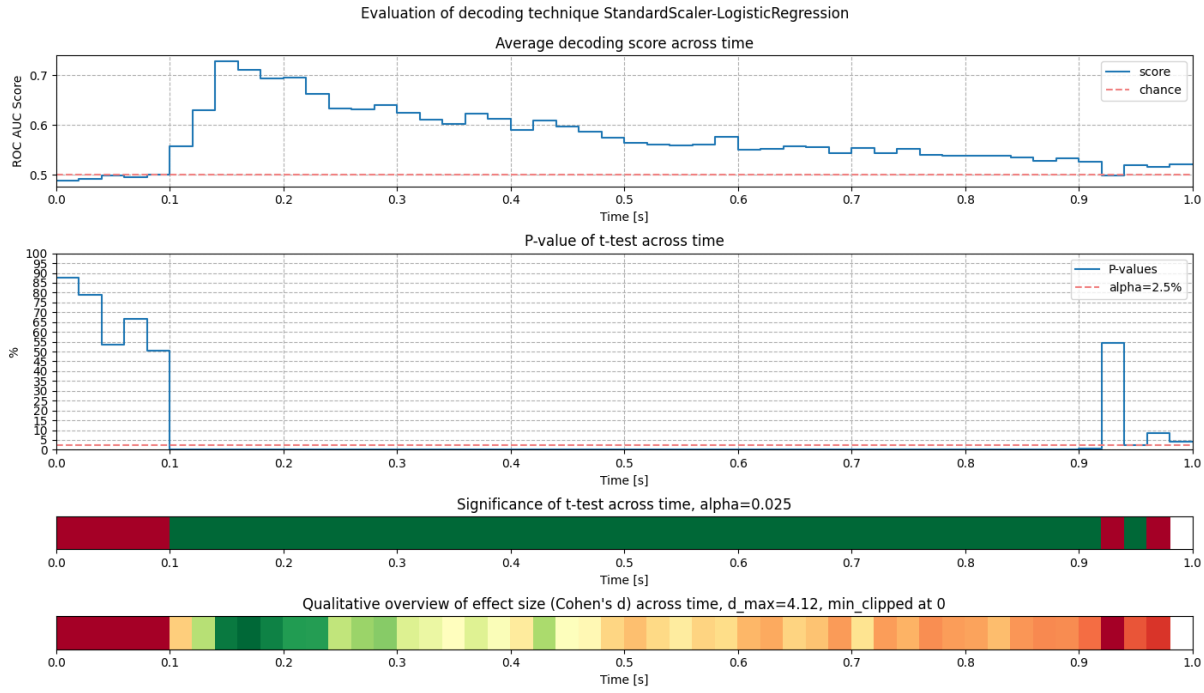
4.2.6 Result Plots

The analysis per scoring strategy is carried out in the following. The conclusions have been drawn in previous sections. It is worthwhile to note that the results are pretty equal for all strategies. This observation supports the quality of the results, because the same scoring could be reached with different feature transformations and different classificators.

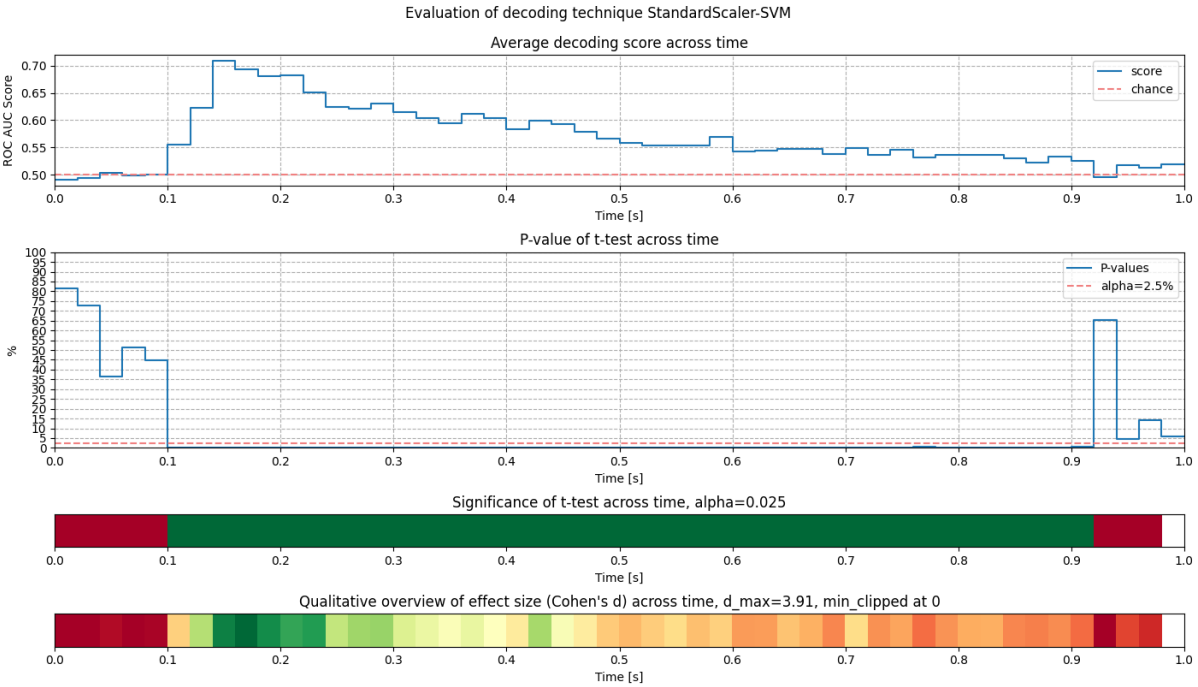
Pipeline: Standard Scaler – Linear Discriminant Analysis



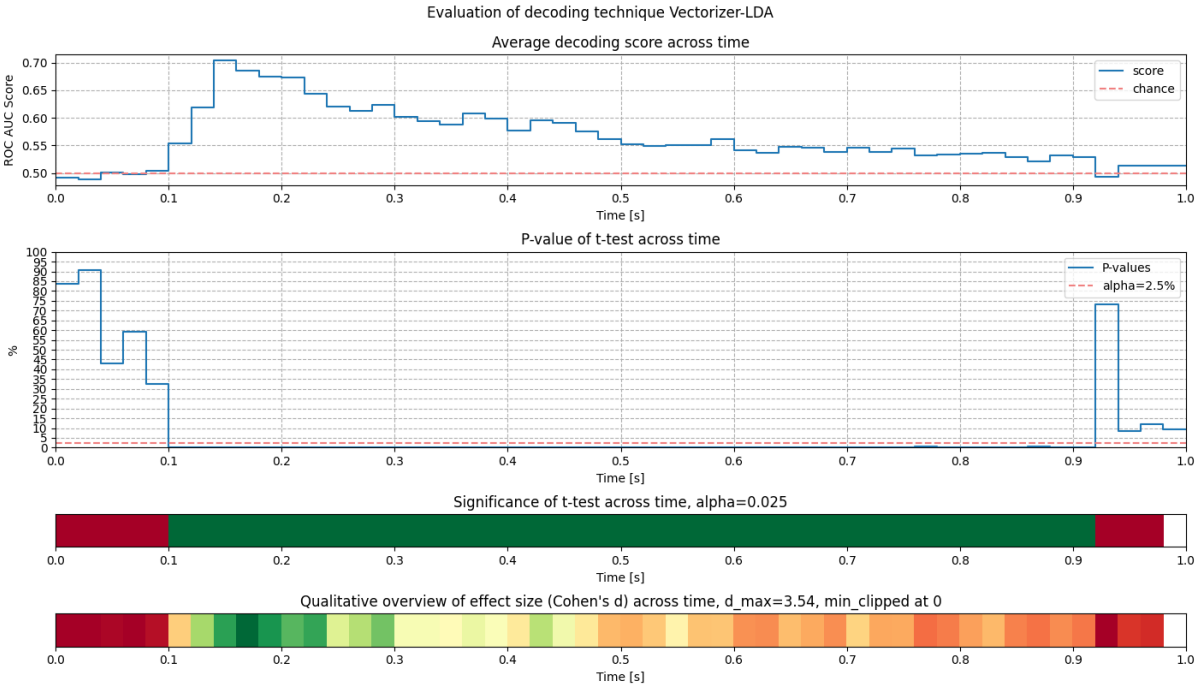
Pipeline: Standard Scaler – Logistic Regression



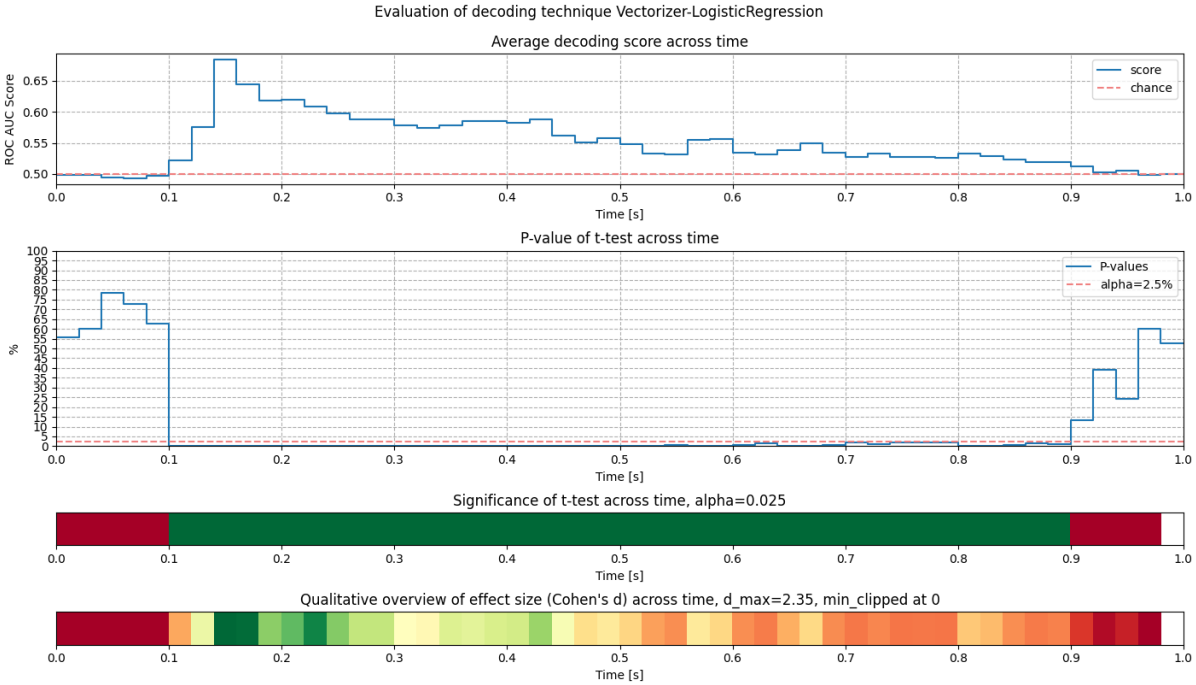
Pipeline: Standard Scaler – Support Vector Machine



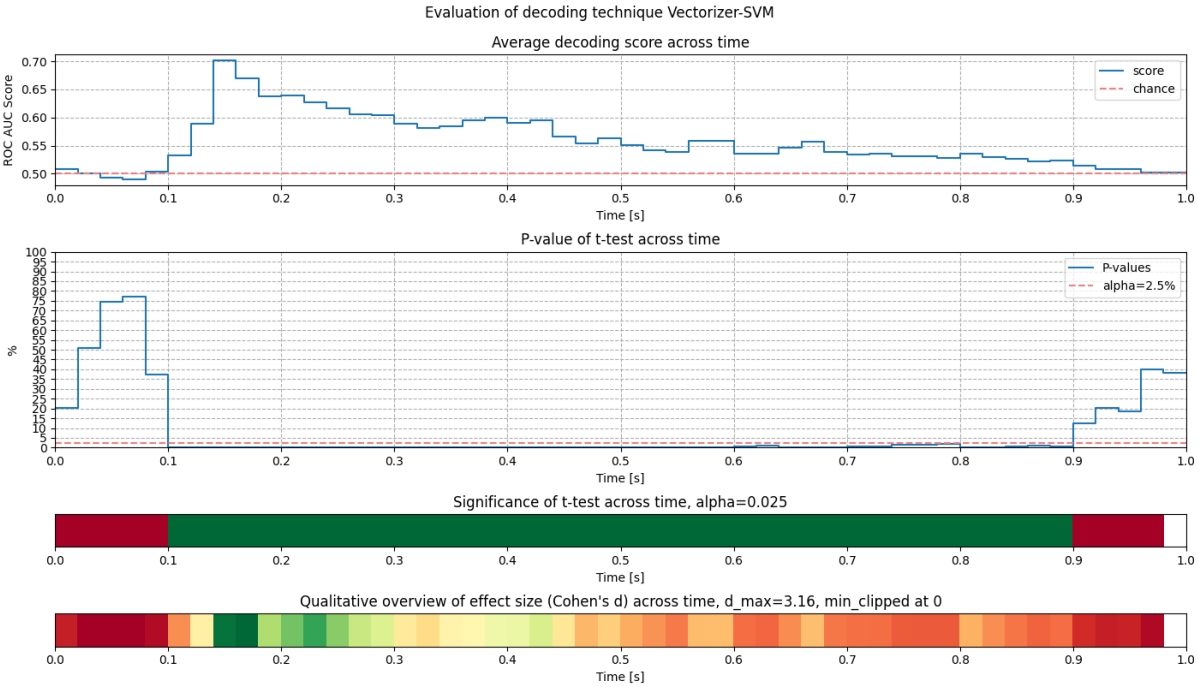
Pipeline: Vectorizer – Linear Discriminant Analysis



Pipeline: Vectorizer – Logistic Regression



Pipeline: Vectorizer – Support Vector Machine



4.3 Time-Frequency Analysis

Files: _12_time_frequency_extraction.py, _22_time_frequency_analysis.py

The goal of this time frequency analysis is to calculate an induced time-frequency analysis of the main experimental contrast. The research question is: What oscillations underlay our effect of interest?

The idea of this work is to split the total activity into two parts. The phase locked part is reflected in the event related potentials, because the timings are clearly defined in relation to the stimulus onset. Hence, the phase locked part is the evoked activity. The remaining activity is non phase locked.

4.3.1 Approach

The analysis is done for PO8, since the N170 effect is strongly measured at this location according to Rossion [24].

The source code of _12_time_frequency_extraction.py describes the procedure per subject:

- Calculate induced component by subtracting evoked from total for conditions “faces” and “cars”
- Transformation of total and induced signals to time-frequency domain using Morlet wavelets for conditions “faces” and “cars”. This step is the key idea of analyzing the induced effect. Averaging on a non phase locked signal in time domain is not possible since the random-phase signals would cause destructive inference and hence cancel out.
- Calculate evoked in time-frequency domain by subtracting total - induced for condition “faces” and “cars”
- Calculate difference power spectrum for total, evoked and induced
- Store

The source code of _22_time_frequency_analysis.py describes the procedure across subjects:

- Average difference power spectrum for total, evoked and induced across subjects
- Statistically test average difference power spectrum for total, evoked and induced

4.3.2 Tradeoff between spatial and temporal resolution

Resolution in space and time are in diametrically opposed relation. An increase of resolution in space always comes with a decrease of resolution in time. Hence, selecting proper values come down to a tradeoff, called uncertainty principle of spectral analysis: $\Delta t \, \Delta f \geq 1$

A variable number of cycles was used for the generation of Morlet wavelets with $n_{cycles} = \frac{f}{3}$. The number of cycles reflects the balance between spatial and temporal resolution. For increasing frequencies, the cycle time increases proportionally. This way the resolution in frequency domain can be improved for higher frequencies. While maintaining a good resolution in time at lower frequencies. The proportional factor stated above was found to yield good results in an empirical evaluation.

Sanity-checking of intermediate results is a key action to ensure high data quality and absence of bugs. Sanity checking has been performed all over the project. During sanity-checking the time-frequency transformation for individual subjects, the importance of those sanity-checks once again

became clear. In the N170 dataset the difference wave has a significant effect starting at around 150-200ms, as shown in the chapter “ERP Peak Analysis”. Hence, the differences of frequency power spectra are expected to show a high power around this time. This was the case for most of the subjects. Few subjects however indicated anomalies. The left part of figure 35 shows subject 002 with the inverse of what was expected, a very low relative power around 200ms. During sanity-checking, this anomaly caught attention and was investigated in detail. Checking the difference plot in temporal domain (see figure 34) made clear that the measured potential at electrode PO8 for subject 002 at the N170 peak is very close for the conditions “faces” and “cars”. In previous processing steps, this was not an issue since calculating the difference in temporal domain has very high temporal precision. In this step, however, the signal was transformed to frequency domain first and the subtraction was performed there. Due to the tradeoff between time and frequency, initially a low time resolution was chosen. As a result, the result of subtraction with low temporal resolution brought cancellation of the signal, resulting in very low power around 200ms. The hypothesis could be confirmed after increasing the temporal resolution. The right side of figure 35 show the result, which clearly shows two points in time (~170ms, ~200ms), that have high relative power. The reason is that at those two points in time the signal in temporal domain changes its slope, which corresponds to activity in the frequency domain. While for most of the time the behavior of the curves match, they traverse their turning points at slightly different points in time. Exactly then the difference of power spectra captures a high relative power signal. This, however, can only be made visible with sufficient temporal resolution. As an outcome of this sanity-check, parameters were adapted to increase the temporal resolution at the cost of reduced frequency resolution.

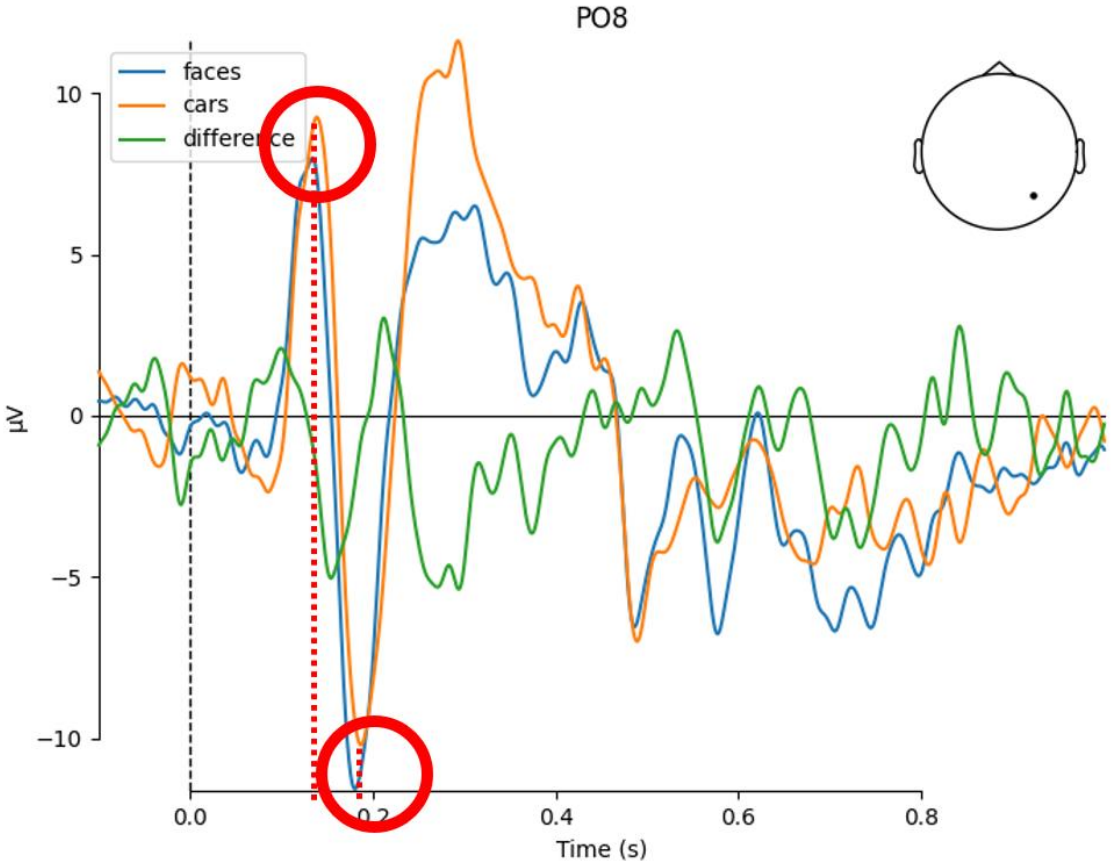


Figure 34: Difference plot between faces and cars conditions for subject 002. The graphs are very closely overlapping. Marked are two turning points around N170 that result in different frequencies at ~170ms and ~200ms.

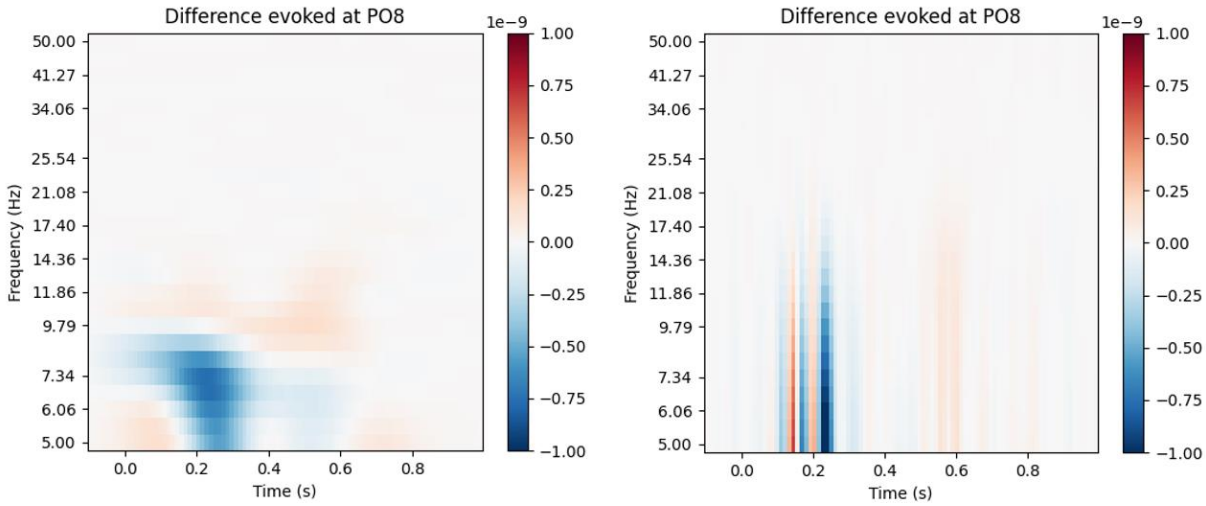


Figure 35: Frequency power spectrum for subject 002. Plotted is the difference between faces and cars of the evoked component. Different cycle numbers have been used for the Morlet wavelet generation to change the resolution in time: cycles = f/1.5 (left) and cycles = f/8 (right).

4.3.3 Power spectrum in time-frequency

Following the approach described before, a power spectrum of the difference wave is generated for total, evoked and induced. Figure 36 gives an overview of the power spectrum of the difference wave in total (left), for the evoked component (center) and for the induced component (right). It can be seen that the total is the sum of the evoked and the induced component. Furthermore, the evoked component has strong power at around 150-200ms, which corresponds to the assumption of N170 effect. The induced component has strong power at around 600-800ms. This makes sense as well because after the phase locked activity triggered by ERP events, with some delay the induced, non phase locked activity is expected.

It is interesting that in the power plots the evoked part reaches equal or larger activity than the induced part. Usually, the expectation would be that the evoked effect is superseded by a stronger induced effect after some delay. The reason might be that the evoked was subtracted individually per condition. Hence, the power spectrum of the evoked difference is much freer of signal of the induced effect and vice versa.

Regarding frequency analysis, it can be generally stated across all three plots, that the main significant activity occurs around 7Hz-14Hz. This matches the frequency band of alpha waves, which are assumed to be correlated with activity of the visual cortex. This observation in general adds plausibility to the time-frequency analysis provided here.

The results have been sanity-checked under further aspects, with regards to causality and plausibility. First, causality is met because the evoked effect occurs and is significant at around 160ms-210ms and the subsequent induced effect is visible and significant at around 600ms-900ms. The results are plausible in a sense that no activation is recognized before around 100ms, which suits to the understanding in literature

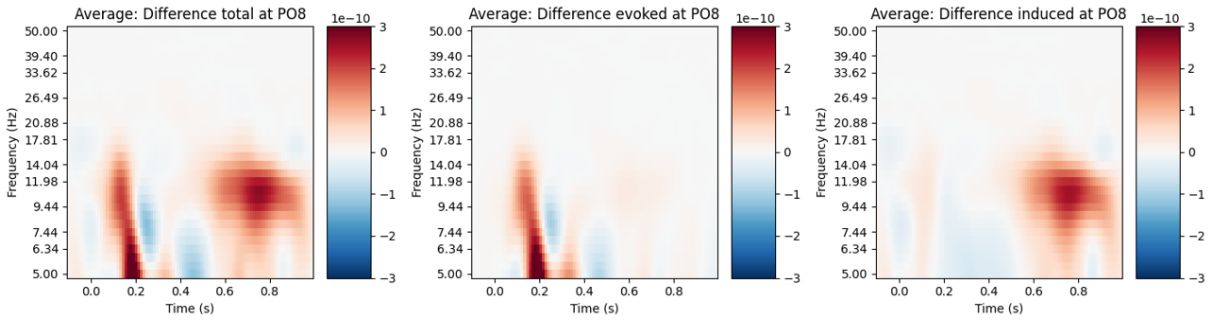


Figure 36: Comparison between power spectrum of difference wave for total, evoked and induced component, averaged across all subjects.

An important part of the procedure is to sanity-check intermediate results in order to ensure correct processing. As all across the project, sanity-checking was heavily applied in this analysis part as well. Most observations have matched the expected outcome. The expected outcome is an intuitive result of a good domain-understanding and helps a lot to check plausibility of the results. When calculating the last processing step, the difference of power spectrum is averaged over all subjects. At this point, a plot is expected that consists of two relatively high clusters in the performance spectrum. One around the N170 peak, which is the evoked part, and one at a later point (600ms-900ms) corresponding to the induced part. Both parts were visible in the plot as expected, see figure 37 on the left. However, additionally another area was visible around $t = 0s$. Checking for each subject individually revealed the problem: Subject 029 apparently is responsible for this cluster around $t = 0s$, since activity around this time is only visible in this subject. A detailed analysis is carried out in section “Bad subjects”, where an outlier epoch has been identified. After removing subject 029 from the list of subjects to process, the power spectrum looks plausible again, as shown in figure 37 on the right. This observation once again emphasizes the importance of regular sanity-checks in the pipeline.

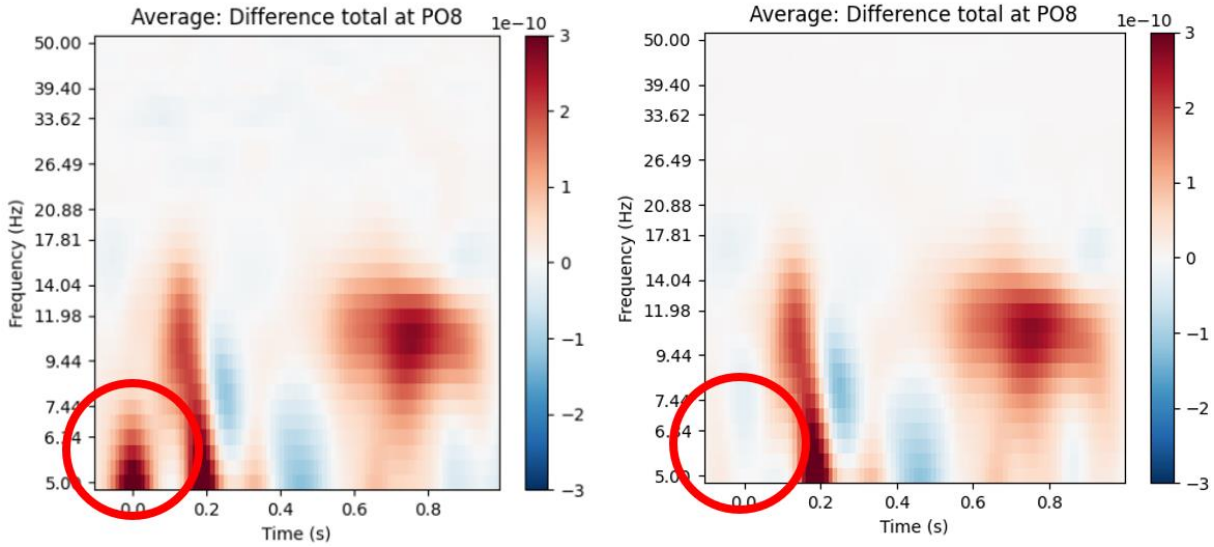


Figure 37: Average activity in power spectrum at t=0s. Left including subject 029, right without subject 029.

4.3.4 Significance testing

To test the resulting power spectrum for significance, a cluster permutation test is chosen [19]. The test is applied three times, for each of total, evoked and induced component. A significance value of 0.05 is chosen and a threshold of 4 for the cluster permutation test was empirically found to work well. The test returns a p-value per cluster of the power spectrum. A cluster can be mapped to a set of tiles in the time-frequency spectrum, in the following denoted with $\Delta t, \Delta f$, where Δt represents the time band and Δf represents the frequency band. The hypotheses are constituted as follows:

$\mathrm{H}_0(\Delta t, \Delta f)$: $power_{difference\_wave}(\Delta t, \Delta f) = 0.0$ i.e., the power of the difference wave has value 0, no difference in specific oscillations Δf at specific time Δt occur when comparing between conditions, no information about the condition is available in this time-frequency band.

$\mathrm{H}_1(\Delta t, \Delta f)$: $power_{difference\_wave}(\Delta t, \Delta f) != 0.0$ i.e., the power of the difference wave has value different from 0, difference in specific oscillations Δf at specific time Δt occur when comparing between conditions, information about the condition is available in this time-frequency band.

Please be aware that 0 does not correspond to no power at all. This scale is logarithmic, hence negative values are actually negative exponents and correspond to small values close to 0. Positive values correspond to large values. It is tested here, whether the power spectrum is significantly different from 0, meaning if significant additional oscillations or significant reduced oscillations occur.

The cluster permutation test gives p-values per cluster, which are mapped back to the time-frequency spectrum $\Delta t, \Delta f$. The total spectrum of power difference is tested as well as the evoked component and the induced component. All three plots are shown in figure 38, where white regions show significant clusters. Compared to the power spectrum in figure 37, it can be seen that the significant clusters match the clusters with high power from the power spectrum.

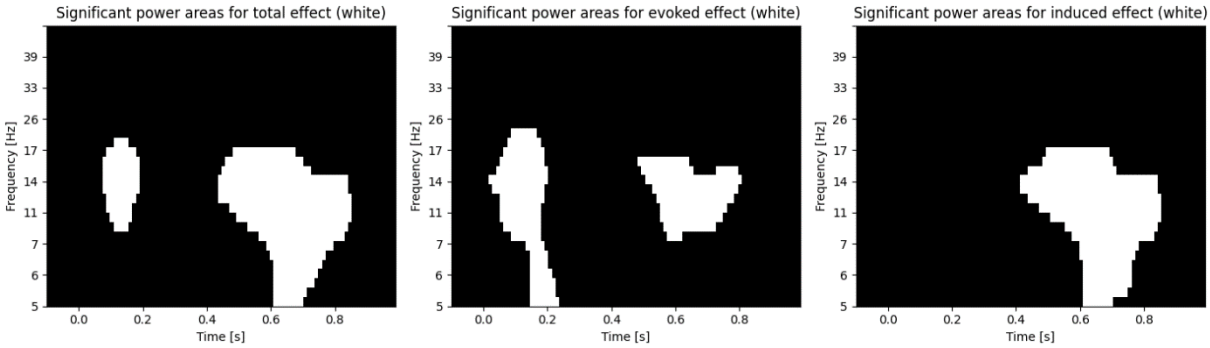


Figure 38: Significance test of difference wave in power spectrum for total, evoked and induced component.

First, the induced component is analyzed. The power spectrum has its main activity between 500ms and 900ms. This matches the expectation that the non phase locked part is triggered with a delay after the ERP-triggered phase locked part.

Secondly, the evoked component is analyzed. Significant clusters are found at around 100-200ms. The evoked component covers phase locked activity, in this case mainly introduced by ERP events. Hence, this time range matches the previous analysis of N170 effect in time. On the other side, in the area of 600-600ms, another significant cluster can be found in the evoked component. This might be the remainder of overlapping non phase locked activity, which is captured in the evoked signal.

The significance of the total power spectrum is roughly a superset of the evoked and induced components, which also makes sense. For completion, it must be noted that the cluster at 150ms is not constantly detected by the cluster permutation test. This might be due to the characteristic of non-deterministic tests and it might be worth analyzing in future work.

5 References

[1] Kappenman, E. S., Farrens, J. L., Zhang, W., Stewart, A. X., & Luck, S. J. (2021). ERP CORE: An open resource for human event-related potential research. *NeuroImage*, *225*, 117465. https://doi.org/10.1016/j.neuroimage.2020.117465

[2] van Vliet, M. (2020). Seven quick tips for analysis scripts in neuroimaging. *PLOS Computational Biology*, *16*(3), e1007358. https://doi.org/10.1371/journal.pcbi.1007358

[3] 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.

[4] *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

[5] *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

[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

[6] Winkler, I., Debener, S., Muller, K. R., & Tangermann, M. (2015). On the influence of high-pass filtering on ICA-based artifact reduction in EEG-ERP. *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, *2015-November*, 4101–4105. https://doi.org/10.1109/EMBC.2015.7319296

[7] Kappenman, E. S., Farrens, J. L., Zhang, W., Stewart, A. X., & Luck, S. J. (2021). ERP CORE: An open resource for human event-related potential research. *NeuroImage*, *225*, 117465. https://doi.org/10.1016/j.neuroimage.2020.117465

[8] *Background information on filtering — MNE 0.22.0 documentation*. (n.d.). Retrieved March 29, 2021, from https://mne.tools/stable/auto_tutorials/discussions/plot_background_filtering.html#defaults-in-mne-python

[9] *Filtering and resampling data — MNE 0.22.0 documentation*. (n.d.). Retrieved March 29, 2021, from https://mne.tools/stable/auto_tutorials/preprocessing/plot_30_filtering_resampling.html#tut-filter-resample

[10] Widmann, A., Schröger, E., & Maess, B. (2015). Digital filter design for electrophysiological data - a practical approach. *Journal of Neuroscience Methods*, *250*, 34–46. https://doi.org/10.1016/j.jneumeth.2014.08.002

[11] Acunzo, D. J., MacKenzie, G., & van Rossum, M. C. W. (2012). Systematic biases in early ERP and ERF components as a result of high-pass filtering. *Journal of Neuroscience Methods*, *209*(1), 212–218. https://doi.org/10.1016/j.jneumeth.2012.06.011

[12] VanRullen, R. (2011). Four common conceptual fallacies in mapping the time course of recognition. In *Frontiers in Psychology* (Vol. 2, Issue DEC). Front Psychol. https://doi.org/10.3389/fpsyg.2011.00365

[13] Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning* (Vol. 1, No. 2). Cambridge: MIT press.

[14] *OSF | N170_Event_Code_Scheme.xlsx*. (n.d.). Retrieved March 29, 2021, from https://osf.io/u8w69/

[15] Luck, S. J. (2014). *An introduction to the event-related potential technique*. MIT press.

[16] *mne.Epochs — MNE 0.22.0 documentation*. (n.d.). Retrieved March 30, 2021, from https://mne.tools/stable/generated/mne.Epochs.html

[17] *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

[18] *OSF | README_N170.txt*. (n.d.). Retrieved March 30, 2021, from https://osf.io/9hd6t/

[19] Ehinger, B. (2016). *Statistics: Cluster Permutation Test – Blog/Science*. https://benediktehinger.de/blog/science/statistics-cluster-permutation-test/

[20] Ehinger, B. (2021). *Why Robust Statistics? – Blog/Science*. https://benediktehinger.de/blog/science/why-robust-statistics/

[21] van Vliet, M. (n.d.). *conpy/fnames.py at master · AaltoImagingLanguage/conpy*. Retrieved March 30, 2021, from https://github.com/AaltoImagingLanguage/conpy/blob/master/scripts/fnames.py

[22] Cohen, M. X. (n.d.). *Overview of possible preprocessing steps - YouTube*. Retrieved March 30, 2021, from https://www.youtube.com/watch?v=JMB9nZNGVyk

[23] *OSF | N170_Subject_Summary.xlsx*. (n.d.). Retrieved March 31, 2021, from https://osf.io/uyb3t/

[24] Jacques, C., d'Arripe, O., & Rossion, B. (2007). The time course of the inversion effect during individual face discrimination. *Journal of Vision*, *7*(8), 3–3. https://doi.org/10.1167/7.8.3

[25] University of California San Diego. (n.d.). *SCCN: Independent Component Labeling*. Retrieved March 31, 2021, from https://labeling.ucsd.edu/tutorial/labels