Published in A Beginner's Guide to Brain-Computer Interfaces

Tim de Boer Follow

Sep 5, 2022 · 5 min read · ✦ · Listen

An Overview Of Outlier Detection Algorithms In EEG Data With Code Examples

A Beginner's Guide to Brain-Computer Interfaces (part 7)

After first talking about using Common Average Referencing as a pre-processing step for EEG data, we

now will talk about another important step in pre-processing: outlier/artifact detection!

This article presents what outliers or artifacts are and how they originate, how they can be manually detected, and which algorithms can be used to automatically detect the artifacts, together with code examples in Python. Enjoy!

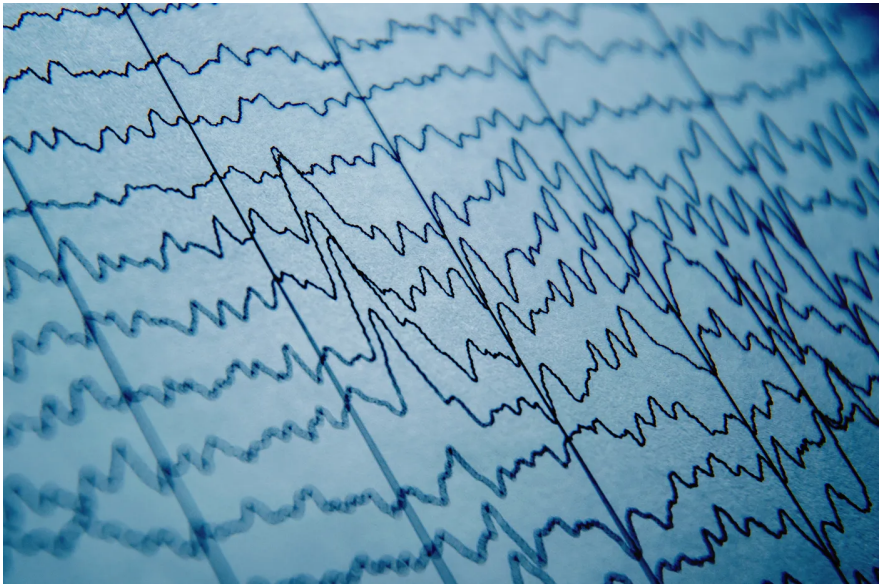

Figure 1: Amplitude peaks in EEG data: artifacts or sudden brain activity?

EEG electrodes can pick up all sort of noise. Electrical noise from other devices in the room, noise from muscle activation signals of eye- or neck muscles, and

noise due to movement of the electrodes or the patient. Basically, any signal not originating from the brain can be called an artifact, or outlier.

Artifacts can make the EEG signal completely unreadable. For example, when a person is clenching his jaw, a high frequency artifact appears in the data, looks like the data in Figure 2. These segments (small windows of the data, often 1 or 2 seconds) of data can easily be seen by the naked eye, and can be manually detected.

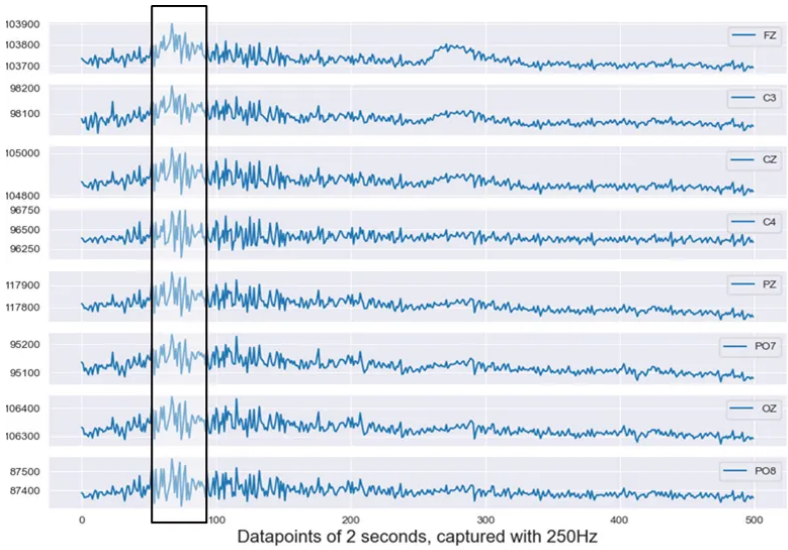

Figure 2: Jaw clenching causes a high frequency noise artifact.

In a review regarding deep learning for BCIs, it was reported that in 29% of investigated papers, a manual approach of detecting artifacts was used. As is it easy to manually spot big artifacts or loss of signal, manual removal of those channels is indeed possible and justifiable for these cases (think of the example shown in Figure 2).

However, noise can also be small, and sparsely present. Because this noise can be very subtle, one may get confused between seeing actual brain activity changes, or seeing noise. Manual detection of these outliers can be very difficult. Moreover, for real-time applications, there simply is no time for a researcher to manually detect outliers. Therefore, automatic artifact detection algorithms have been developed, mostly used to reject segment with artifacts. But even more advanced algorithms have been developed to not only detect artifact, but also delete the artifact and reconstruct the original signal.

Let's go over both approaches in more detail.

Automatic detection & rejection

A paper in 2016 proposed an automatic artifact removal method based on a statistical analysis of the

segments. A segment was considered to contain an artifact when the power, standard deviation, or amplitude of a trial was 2.5 times higher than the mean signal value of all segments, and such segments were rejected. This statistical approach has been adapted in a later study with slight variations to work with real-time experiments by only analyzing the current segment, and not all segments.

Segments will be discarded when the absolute amplitude of the segment would exceed 125 μV, or when the kurtosis of a segment would exceed the standard deviation of the segment by four times.

The authors have claimed the method to be successful, although no comparison was provided between this method and other artifact removal methods. Due to the simplicity of the method, another benefit when compared to more advanced algorithms is the fact that processing time is short, making it suitable for real-time processing. Processing time of other methods can take more time, making it a possible bottleneck for real-time processing.

Coding up this method is relatively simple. Having our EEG data in a Pandas DataFrame as (channels,

timepoints), we can iterate over the rows (channels), and for each channel apply the method. In the example below, the segment is considered an outlier when the method applies to 1 or more of the channels.

```
from scipy import stats
import pandas as pd
import numpy as np

for _, j in segment.iterrows():
   if stats.kurtosis(j) > 4*np.std(j) or
(abs(j - np.mean(j)) >
      125).any():
      outlier = True
```

Automatic detection & Data reconstruction

With Artifact Subspace Reduction (ASR), segments are not rejected, but rather the artifact is removed from the signal, and the original signal is reconstructed. This article will present the idea of the algorithm, together with code, but will not give a mathematical explanation.

Before starting an experiment, the ASR algorithm needs at least 60 seconds of EEG data during which the subject is relaxed and does not move. From this data, ASR is calibrated by learning the space of the subject

and session specific EEG data, by computing the covariance matrix.

During the experiment, the calibrated algorithm will go over each segment to check whether the segment contains artifacts. If artifacts are found, the large-variance components in the segments are removed. Then, using the correlation structure of the calibration data and the remaining components in the experiment data, the "clean" data can be reconstructed.

To use ASR in Python, we can use the MEEGkit package:

```
pip install
git+https://github.com/nbara/python-
meegkit.git
```

Then, we can import the ASR algorithm, and calibrate it using the 60 seconds of calibration data we captured. An important thing to note here it that the calibration data may not contain low frequency signals, thus should be high pass filtered (above 0.5 Hz, or 1 Hz) before applying the ASR algorithm. Our data is in a (channels, timepoints) format.

```
from meegkit.asr import ASR
from scipy import signal

asr = ASR(method='euclid')

calibrate_data =
pd.read_csv(calibrate_data_path)
sample_freq = 250 #250Hz
b_env, a_env = signal.butter(2,
0.5/(sample_freq/2), 'highpass')
caldata_filt = signal.filtfilt(b_env,
a_env, calibrate_data)

asr = ASR(method='euclid')
asr.fit(caldata_filt)
```

Now, ASR is ready to be applied to the segments of our experiment data. Our data was first segmented (in my case, I first segmented my big DataFrame into DataFrames of 2 second segment). The ASR algorithm accepts Numpy, so before we apply our algorithm, we transform the segment into a Numpy array. Sometimes, ASR did not work, so I build a try except argument around it.

```
# for each 0.5 to 2 second segment of EEG
data:
try:
   # print('applying ASR...')
   curr_segment =
```

```
asr.transform(curr_segment.to_numpy())
except:
   noASR += 1
   print(f'didnt apply ASR for {noASR}
times out of total of
   {total_seg} segments.')
```

Conclusion

This article explained what artifacts in EEG data are, how they originate, and presented algorithms including code examples in Python to automatically detect these artifacts.

It is important to know that these algorithms are not perfect, and taking measures to prevent artifacts beforehand is always a smart thing to do. For example, remind participants of the nature of EEG and ask them to avoid movements or actions that might contaminate the signals as much as possible. Think of excessive eye blinking, moving their head, jaw clenching. These things may happen subconsciously, so for the researchers it is also important to monitor what the subject is doing.

Of course the presented algorithms are not the only ones out there. For example, other popular methods are ICA, and the h-∞ algorithm. Nevertheless, I hope

this article presented a clear introduction into the field of artifact detection for EEG data. During my literature search, I found another blog post from BitBrain which gives an excellent, more in-depth overview of artifacts in EEG, and may serve as a next step into understanding this field.

Please check out the publication page, where more practical BCI tutorials like this one are already posted, and will be posted in the future, and give me a follow to be notified for further posts!

Brain Computer Interface Outlier Detection Eeg

Python Bci

Get an email whenever Tim de Boer publishes.

Your email

We couldn't process your request. Try again, or contact our support team.

Subscribe

By signing up, you will create a Medium account if you don't already have one. Review our Privacy Policy for more information about our privacy practices.