EEG Preprocessing

Steps to preprocess EEG data generally include the following:

1. Importing the raw data
2. Downsample the data
3. Bandpass filter
4. Re-reference data
5. Inspect electrodes and reject noisy channels
6. Epoch the data
7. Inspect and reject noisy epochs
8. Run independent component analysis and reject noisy components
9. Save preprocessed data

These steps are sometimes done in a different order, or some of the steps omitted depending on a researcher's preference.

The Python package we use to run through each of these steps is MNE: [https://mne.tools/stable/index.html](https://mne.tools/stable/index.html)

Installation instructions can be found [here](https://mne.tools/stable/installation.html) and is made relatively simple with Anaconda.

Below I will go into further detail on how to use MNE for the above preprocessing steps. However, you may refer to [this website from Mike X Cohen](https://mike-cohen.github.io/EEGinction/) (navigate to the left side of the webpage > Data Pre-processing and cleaning) for further information as to the rationale behind each step as well as other instructional videos for EEG data analyses.

**Importing the raw data**

To import the raw data, first locate the directory in which the raw data is stored (should be a sub-directory within the RDSS). Then, use the function `mne.io.read_raw_bdf()` to read the data into an MNE Raw object.

*Pay attention* to some of the deprecation warnings on these webpages, as some of the specifics of the commands used here may be outdated depending on the version of MNE you are using. This tutorial is using MNE version 0.18.2. This info can be found by using the command `mne.sys_info()`.

Here, we need to specify the montage for MNE to use, i.e. the coordinate information for the type of electrode cap used. This can be defined using `mne.channels.read_montage()`.

Upon import, you can plot the data to get a quick sense of how reasonable it looks.
MNE will try to automatically adjust the scale of the data when plotting each electrode, but sometimes this feature fails and results in an odd-looking plot, such as below:

The scalings parameter in raw.plot() can be adjusted to display the desired scale of the data.

Once the raw data is plotted, it can be scrolled through interactively to review each channel for noise.

**Bandpass Filtering the Data**

The data needs to be filtered for low-frequency and high-frequency signal, which is often resultant from environmental/muscle noise in scalp EEG and otherwise is not generally the focus of analyses. Low-pass and high-pass filtering allows for noise below and above a certain frequency to remain in the data.

This can done using the MNE command raw.filter(), and you must specify what you want your band cut-offs to be. Typically, and depending on your planned analyses, filtering will be set around 1 and 50 Hertz.

Bandpass filtering will also have the effect of smoothing out the raw data, and typically looks different than raw data to the naked eye.

**Re-referencing**

Re-referencing also helps clean the data by providing an estimate of baseline activity of physiological noise. Typically, the reference electrodes will be one of the external electrodes (in the case that you’re using the 64 Ch Biosemi system). I use the two mastoid channels. Re-referencing is one reason why it is very important to know which external electrodes were plugged into which channel, and to keep this consistent between subjects.

**Inspecting channels**

Noisy channels can be rejected and interpolated. There are functions to automate this process, but I prefer to visually inspect them. Channels that are marked for rejection are typically noisy throughout the continuous data. Don't reject channels that are only periodically noisy, mostly because periods of noise typically affect most or all channels all at once, and also because brief periods of noise can be rejected in the epoching stage. You can mark channels for interpolation by selecting the channel label on the left side of the plot.
ie, here you may decide that P7 is too noisy throughout the file. You can scroll through all of the time points to help determine this by clicking your left and right arrows on the keyboard.

**Epoching**

The data should be epoched based on the different stages in a trial. This step of preprocessing is why it is so vital that we ensure accurate timing in sending triggers from our Psychopy script to ActiveView (the EEG recording software from Biosemi). The reason why we epoch data is to have discrete time periods during which we know which X was happening on the screen and therefore we suspect that Y cognitive process may be reflected in the data, and as such we want to narrow our analyses to this time period of each trial (and possibly compile across the trials).

For example, say we have a simple flanker task where arrows pop up on the screen which are either congruent (ie pointing in the same direction <<< ) or incongruent (pointing in different directions <>< ). Congruent and incongruent arrows are our two conditions. Say the arrows are presented on the screen for 1000 milliseconds, during which a subject must indicate the direction of the center arrow, and then an inter-trial interval of 3400-5000 ms (jittered) follows. Our triggers for the incongruent condition is 101, congruent is 103, and these triggers are sent precisely on the onset of the arrows to the screen.

When you epoch the continuous data using MNE, it is constructed into a new **MNE Epochs** object. This Epochs object requires you to specify the trigger for the trial period in which you are interested, as well as the baseline period to use and the continuous file to be epoched.

In this case, defining our epochs may look something like this:

```python
cue_epochs = mne.Epochs( raw=raw, events=events, event_id={'Incongruent':101,'Congruent':103}, tmin= -0.8, tmax=1.0, baseline=(None,-0.3), metadata= this_subjects_behavioral_data)
```

- Where "raw" is an example variable name which refers to the continuous data file you've been editing up to this point.
- "Events" are the triggers detected throughout the whole bdf (can be generated using the function mne.find_events( ) ). "Event_id" are the trigger numbers that correspond to this period of the trial. Notice that we are feeding in both conditions to one Epochs object. This is because we can easily index each of these conditions later on using the MNE Epochs object.
- The "tmin" and "tmax" parameters tell MNE when to slice the continuous data around the given triggers. Here, I've set tmin to 800 ms before the onset of the arrows and the tmax to be the end of the arrows' presentation on screen.
- Why do we set the tmin to be before the beginning of the trigger? This is to allow our Epochs to be baseline corrected to the ITI, meaning that we have some ability to set a reference for the "baseline" neural/noisy activity that is consistently present in the data. By baseline correcting, we're more easily able to pull out neural activity that is directly related to the condition of interest. This is why we set "baseline" to be (None, -0.3). This tells MNE to use the tmin as the beginning of the baseline period, up to -300 ms. We don't use the whole period from -800 ms to 0 ms because this can cause problems in our time-frequency analyses later on.
- The "metadata" parameter is expecting a Pandas dataframe of the same length as there are events (ie, the number of trials should match between the behavioral log and the BDF). The dataframe should contain info about each trial, including the behavioral data such as reaction time and accuracy. It is helpful to assign the metadata here, upon creation of the epochs, as it enables you to keep track of which trials were rejected during inspection of the EEG data.

**Inspecting Epochs**

Epochs can be interactively scrolled through when using an iPython session or running a python script. MNE allows for you to select epochs for rejection, and automatically marks those as 'bad' epochs upon closing the figure. These bad epochs can then be dropped.

![ Epochs visualization with MNE ]

**Running ICA**
The next step after cleaning your Epochs is to run it through Independent Component Analysis. This allows you to reject components of the data that seem to be heavily influenced by motor-related artifacts from blinking, jaw, neck, arm, or upper back movement. Inspecting components is perhaps one of the trickiest parts of the pre-processing pipeline for beginners. EEGLab is a Matlab-based package for processing EEG data, but their website is very useful for help with examples of real vs artifact-induced data. Here is their page on detecting IC’s that you’d want to reject (though it is illustrated using the EEGLab GUI, their topographic maps are similar enough to be useful).

The biggest thing to look out for are hotspots of activity around the edges of the topoplot. Blink artifacts are typically easiest to identify using this method, as they appear near the very front of the topomap (around the frontal channels). Jaw artifacts appear at the very sides of the topomap. Otherwise, highly concentrated spots of activity are typically an artifact of one noisy channel.

Notice that some of the labels are automatically greyed out. This is because MNE has the ability to auto-detect which IC’s are a result of muscle noise when given information about EOG and ECG electrodes. It is imperfect (as evidenced here by the 'rejected' ICA002, which I would not flag for rejection) so make sure to not blindly trust it.