Kurzanleitung MNE-Python Pipeline

Installation:

Für die Installation die Anleitung der MNE-Python-Website befolgen:

https://www.martinos.org/mne/stable/install_mne_python.html

Ich habe in die Pipeline autoreject eingebaut. Außerdem ist ein weiteres Package nötig, um Source-Estimates als .mp4 zu speichern. Zu Beginn müssen für diese Pipeline also auch noch dieses Packages installiert werden:

1. Activate mne
2. pip install https://api.github.com/repos/autoreject/autoreject/zipball/master
3. conda install ffmpeg -c conda-forge

Lege die **Pipeline_<Dein Projekt>.py-Datei**, als auch den Ordner **Pipeline_Functions** an den selben Speicherort. Das ist wichtig, damit die Pipeline_Functions gefunden werden können. Diese Skripte können jedoch irgendwo auf dem Rechner abgelegt sein, sie müssen nicht an demselben Ort deiner MEG-Dateien liegen.

Ich arbeite gerne mit **Spyder**, um die Pipeline zu bearbeiten und auszuführen. Dazu im Anaconda Prompt nach „activate mne“ „spyder“ eingeben oder die entrsprechende Verknüpfung benutzen. Atom oder andere IDE's sollten aber auch funktionieren, wenn die Interpreter für das mne-coda-environment eingerichtet sind.

Skript-Struktur:

- Pipeline_<Dein_Projekt>.py ruft auf:
  - subject_organisation.py (Anlage der Dateilisten und Zuordnungen)
  - operations_functions.py (Hauptfunktionen Analyse)
  - plot_functions.py (Plotten der Ergebnisse)
  - Untergeordnet:
    - io_functions.py (import der dateien, hauptsächlich handling der Benamung)
    - utilities.py (verschiedene Unterfunktionen)
    - decorators.py (bisher nur ein Decorator)
    - operations_dict.py (speichert Funktionsnamen für Funktions-GUI)
    - func_cache.py (speicher Auswahl von Funktions-GUI)

Vorbereitung

Ordner einrichten:

1. Bei **home_path** den Pfad für den Ordner auf eurem Computer/Festplatte mit allen Labor-Daten eingeben, jeweils für das passende Betriebssystem, falls du mehrere nutzt
2. Bei **project_name** den Pfad des Projektordners für euer Projekt eingeben (so können mit derselben Pipeline verschieden Projekte analysiert werden)
3. Bei **subjects_dir** den Pfad zum Ordner mit den schon von Freesurfer-segmentierten MRT-Sequenzen eingeben
4. Bei **orig_data_path** den Pfad zu einem Ordner angeben, in dem du die fif-Dateien deiner Messungen in ihrem Datumsordner ablegst (z.B.“170325m1“)

orig_data_path befüllen:

In den Ordner von orig_data_path kommen alle Ordner wie z.B. 170325m1 von deinen Messungen

Erster Start

Nun wird das Pipeline-Skript das erste Mal gestartet (F5 in Spyder). Dabei erscheint ein Fenster, in dem die Funktionen ausgewählt werden können. Beim ersten Starten sollten alle subject_operations ausgewählt werden und sonst keine Funktionen

Dateien einpflegen (Subject Organisation):

Wichtig: Eine einheitliche Benennung der Messdateien macht das Handling sehr viel einfacher. Außerdem sollten die Leerraummessungen das Wort „leer“ im Namen haben, damit sie automatisch erkannt werden können.

1. **Add_files:**

   Diese Funktion fügt die Namen aller fif-Dateien, die sie in den Ordnern im Pfad orig_data_path finden kann, zu sub_list.py hinzu und die Dateien in die entsprechenden Ordner im Projetkordner kopiert. In orig_data_path sollten also die Ordner wie „170325m1“ liegen, die du aus dem NAS20TB-Archiv bekommst. Damit die Leerraummessungen automatisch erkannt werden können, müssen sie „leer“ im Namen haben

2. **Add_mri_subjects:**

   Hier werden die Namen aller Ordner aus subjects_dir automatisch hinzugefügt**.**

3. **Assign Subject to MRI-Subject:**

   Hier werden die Dateinamen der jeweils passenden Freesurfer-Segmentierung zugeordnet. So können z.B die Messungen eines Messtages an derselben Person alle derselben Freesurfer-Segmentierung dieser Person zugeordnet werden

   a. Dateiname einer Messung auswählen
   b. Zugehörige Freesurfer-Segmentierung(Ordner-Name) auswählen
   c. **Assign** drücken

4. **Assign ERM to Subject:**

   Hier werden den Messungen die jeweilige Leerraummessung zugeordnet. Wenn die Leerraummessungen nach einem einheitlichen Schema benannt werden, kann diese Zuteilung auch mit Regular-Expressions vereinfacht werden.

   a. Dateiname (**ohne .fif**) der Leeraummessung auswählen
   b. Zugehörige Messung auswählen (wahrscheinlich verwendet man für die Messungen eines Tages mehrmals dieselbe Leeraummessung)
   c. Wenn keine Leerraummessung existiert, dann sollte **„None“** ausgewählt werden
   d. **Assign** drücken

5. **Assign bad channels to Subject:**
   a. Dateiname auswählen
   b. **“plot_raw”** drücken
   c. Schlechte Kanäle identifizieren
   d. Sie können zum Markieren angeklickt werden. Das reicht jedoch **NICHT**, um sie als bad_channels zu speichern.

e. Dafür muss erst in dem Fenster noch ein **Haken** bei der entsprechenden Zahl gemacht werden.
f. Nachdem alle Haken gemacht wurden **Assign** drücken (Dateiname muss in der Liste ausgewählt sein, Änderung gehen verloren, wenn ein anderer Name ausgewählt wird und nicht Assign gedrück wurde)
g. Grün bedeutet, dass schon einmal bad_channels gespeichert wurden, rot nicht
h. Die Liste der bad_channels kannst du entweder in dem GUI verändern (Haken neu setzen, Assign drücken) oder bad_channels_dict.py direkt in einem Editor bearbeiten **(!Achte generell bei den sub_scripts darauf, keine Leerzeichen oder andere falsche Zeichen einzufügen und die Formatierung zu wahren!)**

Jetzt sollten die MEG-Dateien vorbereitet sein. Fehlt noch die Freesurfer-Segmentierung

Vorbereitung der Freesurfer-Daten:

1. Die MRT-Bilder jedes Probanden sollten von Freesurfer segmentiert worden sein und die Ordner in dem vorher als **subjects_dir** deklarierten Ordner liegen
2. Dann müssen noch die **mri_subject_operations** („apply_watershed“ bis „morph_subject“) durchgeführt werden. Diese funktionieren bisher nur auf einem UNIX-System (nicht Windows!)
3. Dazu bei **which_mri_subject „all“** oder die entsprechende Zeilennummer der gewünschten Freesurfer-Ordner in mri_sub_list.py auswählen
4. In dem Funktions-Fenster die benötigten mri_subject_operations auswählen und dann das Skript ausführen (F5 in Spyder)

Koregistrierung

Um die Koordinatensysteme vom MEG und MRT zusammenzubringen, muss die Koregistrierung durchgeführt werden. Dazu gibt es in MNE-Python eine extra Benutzeroberfläche.

Diese kann entweder über die Funktion mri_coreg aus der Pipeline aufgerufen werden oder unter UNIX-Systemen im aktivierten conda-mne-environment mit „mne coreg“.

Eine Anleitung dazu gibt es hier:

https://www.slideshare.net/mne-python/mnepython-coregistration

Diese Benutzeroberfläche ist leider moment recht anfällig für Abstürze. Bei mir stürzt sie verlässlich ab, wenn ich versucht, das Subject rechts oben zu ändern, sowohl auf Windows als auf Linux. Ich hoffe, dass dieses Tool in Zukunft verlässlicher wird.

(Die Konsole in Spyder muss im Moment bei Verwendung der Funktion mri_coreg nach jedem Schließen des Coreg-Fensters neugestartet werden)

Hat man für eine Messung ein Trans-File erstellt, kann man dieses theoretisch auch für die anderen Messungen desselben Tages (genauer mit derselben Digitalisierung) verwenden. Dazu muss dieses Trans-File jedoch auch **manuell** in die entsprechend anderen File-Ordner im Ordner Daten kopiert werden.

< Vorbereitung fertig >

Analyse

Dateien/Freesurfer-Ordner für Pipeline-Durchgang auswählen:

Vor jedem Durchgang werden bei Which_Subject am Anfang des Pipeline-Skriptes die Zeilennummern der Dateinamen, die analysiert werden sollen, ausgewählt. Dabei können auch Zahlen-Spannen eingegeben werden, die analysiert oder von der Analyse ausgeschlossen werden sollen. Weitere Informationen dazu im Pipeline-Skript.

Parameter einstellen:

Je nach Parameter sind hier Integer, Float, Boolean oder String gefordert. Die Erklärungen stehen oft dabei oder könne durch Verfolgen des Weges der Variablen erschlossen werden.

Funktionen auswählen:

Zunächst sollten die raw-fif-Dateien analyisert werden. Standard-Schritte sind hier:

Sensor_space_operations:

filter_raw – find_events – epoch_raw – run_ica – apply_ica – get_evokeds – tfr

source_space_operations:

create_forward_solution – estimate_noise_covariance – create_inverse_operator – source_estimate

Danach die gewünschten Plots (die entsprechende Analyse muss natürlich jeweils vorher gelaufen sein)

Mehr Personalisierung

Funktionen hinzufügen:

Die Funktionen sollten in operations_functions.py definiert und vom Pipeline-Skript aus mit den entsprechenden Variablen gerufen werden(auf die richtige Reihenfolge der Variablen achten).

Für die Funktion muss außerdem ein zum Namen äquivalenter Eintrag in operations_dict.py eingetragen werden, damit if exec_ops[<Funktions-Name>] keinen Key-Error produziert.

Wenn du eine Funktion in deinem Pipeline-Skript hinzufügt, dann aktualisiere bitte auch das Pipeline.py-Skript mit einem neue branch dafür und pull-request auf GitHub.

Regular Expressions nutzen:

Python bietet mit Regular Expressions ein tolles Tool, um mit einer systematischen Benennung der Messung die größtmögliche Effizienz bei der Analyse zu erzielen, z.B. bei der Einteilung der File-Gruppen für die Grand-Averages.

Eigene Plots erstellen:

Mit matplotlib und PySurfer bzw. mayavi sind der eigenen Kreativität kaum Grenzen gesetzt

Auf GitHub kollaborieren:

Schick mir gerne deinen GitHub-Namen, dann füge ich dich zum Repository für diese Pipeline hinzu. So könnte jeder neue Funktionen oder Plots in eine wachsende Labor-Pipeline integrieren.

Ich hoffe, ich kann mit dieser Pipeline den Einstieg in MNE-Python etwas erleichtern. Die Stärke von MNE-Python ist, dass man sich eine Pipeline sehr genau auf seine Bedürfnisse zuschneiden kann. Die Pipeline ist also nicht als starres Programm gedacht, sondern als Grundstruktur auf der aufbauend hoffentlich ein möglichst individuell passendes Analyse-Tool entsteht. Dabei kommt es jedoch oft auch zu Fehlermeldungen, die man mit etwas Geduld zu ihrem Ursprung zurückverfolgen muss.

Bei Fragen stehe ich gerne zur Verfügung.

martin.schulz@stud.uni-heidelberg.de

Diese Pipeline wurde auf Grundlage von Lau Andersens Pipeline erstellt:

Andersen, L. M. (2018). Group analysis in MNE-python of evoked responses from a tactile stimulation paradigm: A pipeline for reproducibility at every step of processing, going from individual sensor space representations to an across-group source space representation. Frontiers in Neuroscience, 12(JAN). https://doi.org/10.3389/fnins.2018.00006