EEG/MEG Data Analysis Using MNE-Python

Vanessa Hadid and Annalisa Pascarella

MAIN EDUCATIONAL 2024 - Oct 25

Outline

Section 1: Introduction to EEG/MEG Data Analysis Using MNE Python

Section 2: Explore raw data, preprocess it, and generate evoked responses - *script 1*

Section 3: Explore multivariate statistics to decode information from M/EEG data - *script 2*

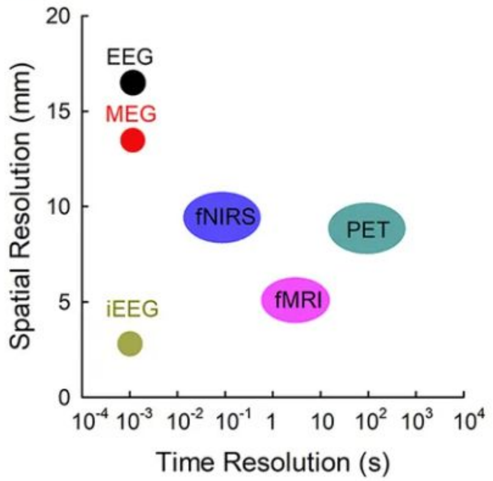
Section 4: Apply the Common Spatial Pattern (CSP) to decode motor imagery from EEG data - *script 3*

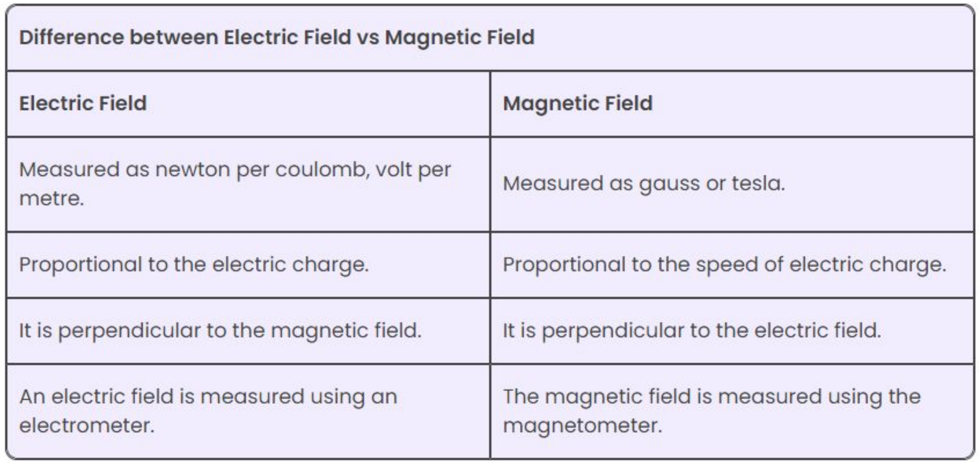
Section 5: Hands-On

Section 1

Introduction to EEG/MEG Data Analysis Using MNE Python

Figure 1

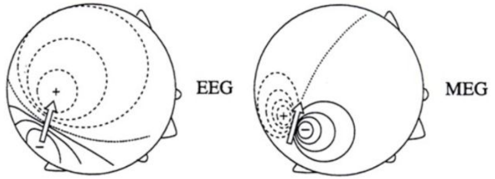


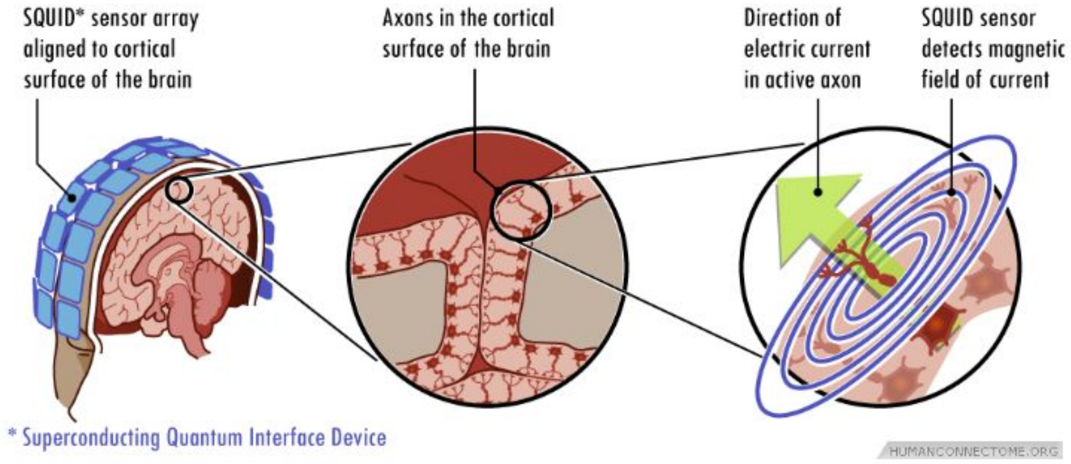


Difference between Electric Field vs Magnetic Field	
Electric Field	**Magnetic Field**
Measured as newton per coulomb, volt per metre.	Measured as gauss or tesla.
Proportional to the electric charge.	Proportional to the speed of electric charge.
It is perpendicular to the magnetic field.	It is perpendicular to the electric field.
An electric field is measured using an electrometer.	The magnetic field is measured using the magnetometer.

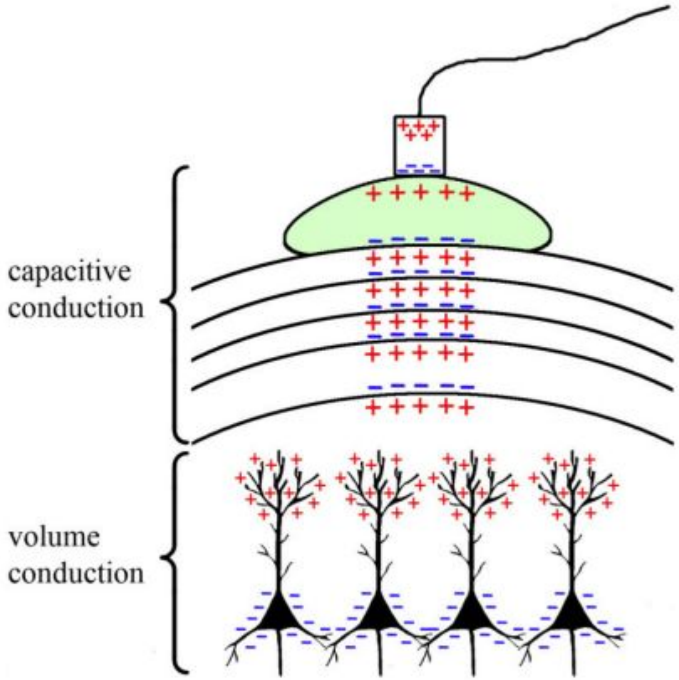
Xu et al., 2021, Frontiers in Human Neuroscience

Introduction to EEG/MEG Data Analysis Using MNE Python



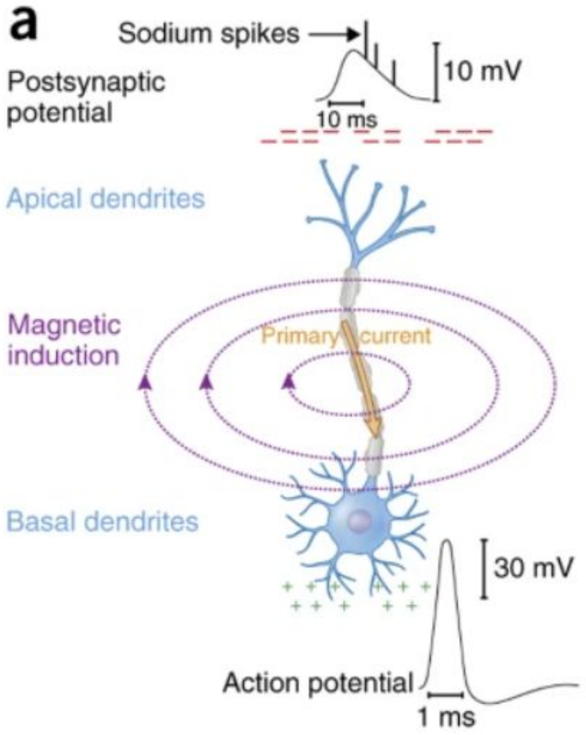

SQUID* sensor array aligned to cortical surface of the brain
Axons in the cortical surface of the brain
Direction of electric current in active axon
SQUID sensor detects magnetic field of current
* Superconducting Quantum Interface Device
HUMANCONNECTOME.ORG

EEG: Electric field



Jackson & Bolger 2014, Psychophysiology

MEG: Magnetic field



Baillet 2017, Nature Neuroscience

Workshop Objectives

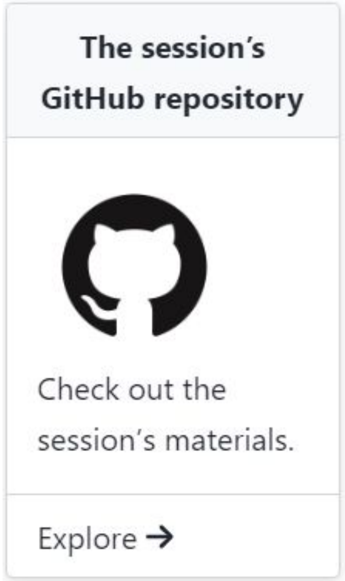
Use the capabilities of MNE-Python to process and analyze EEG and MEG data.

Content that will be covered:

- Preprocessing raw data
- Applying advanced techniques like Multivariate Pattern Analysis (MVPA)
- Applying decoding to motor imagery tasks

Repository : https://github.com/thecocolab/mne_meeg_ml_main

MNE : https://mne.tools/stable/documentation/index.html



The session's GitHub repository

Check out the session's materials.

Explore →

Section 2

From Raw Data to Evoked Responses - ***script 1***

Getting Started with MNE-Python

Setup and Data Handling

Loading the Sample Dataset

Preprocessing and Inspecting Raw Data

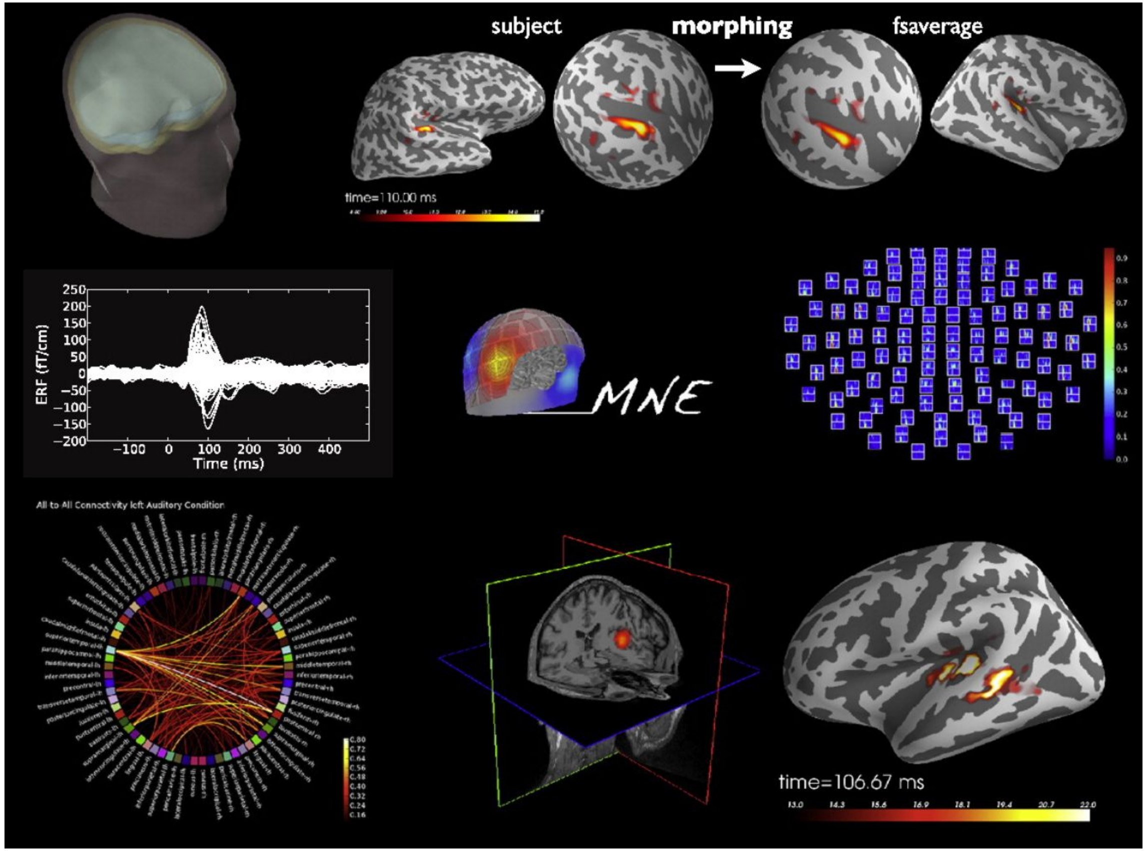
Defining and Extracting Epochs

Averaging Epochs and Visualizing Evoked Responses

Getting Started with MNE-Python

1. **Introduction to MNE-Python**
    - MNE-Python is a powerful open-source Python library specifically designed for the analysis of neurophysiological data like EEG and MEG.
    - It facilitates complex analysis pipelines from preprocessing to advanced statistical analysis.
2. **Key Features**
    - From raw data, preprocessing, and visualization, to time-frequency analysis and source estimation.
    - Integration with machine learning libraries: integrates with scikit-learn for advanced decoding.
    - Community-driven: Regularly updated with new features and comprehensive documentation supported by an active community.
3. **Purpose in Neuroimaging**
    - The primary goal of MNE-Python is to make sophisticated statistical tools accessible for non-programmers while offering maximum flexibility for expert users.
    - It supports the entire workflow in M/EEG research, from the acquisition of raw data to the publication of scientific results.

Getting Started with MNE-Python

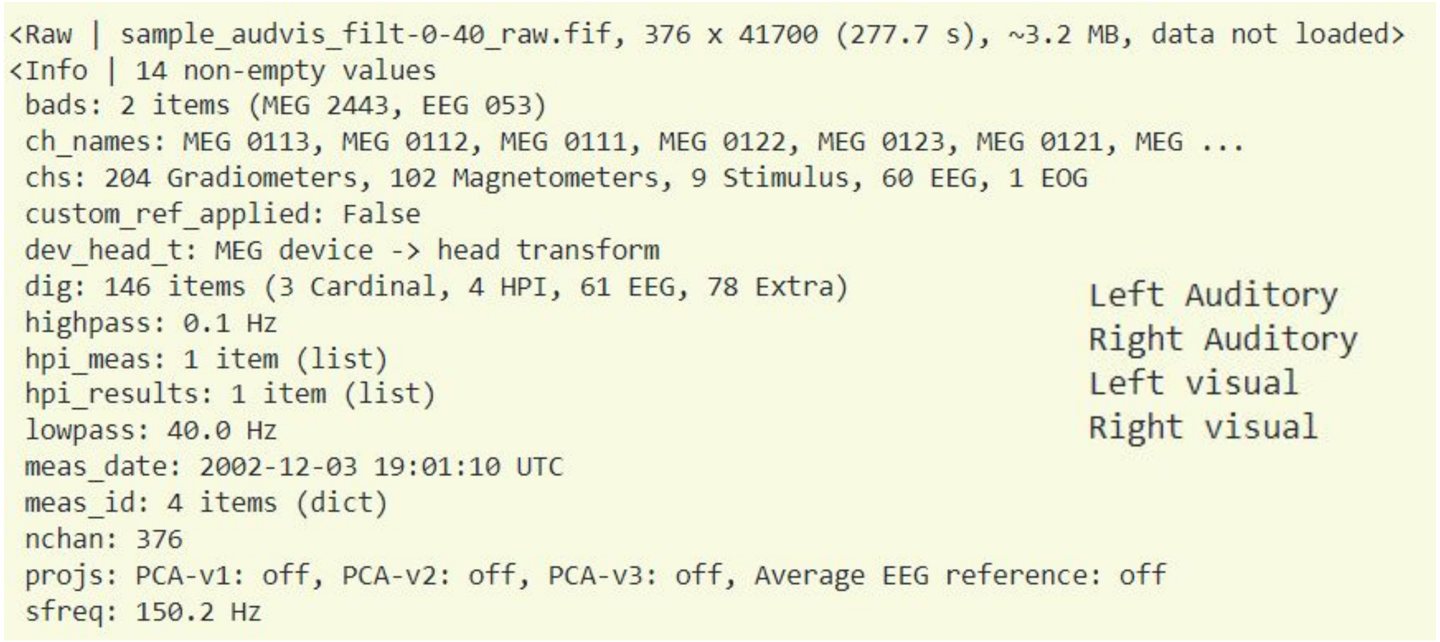


Gramfort et al., 2014, Neuroimage

Setup and Data Handling in MNE-Python

1. **Environment Setup**
   - MNE-Python runs on Python versions 3.6 and above.
   - You can install MNE-Python with the command: `pip install mne` or `conda install -c conda-forge mne`.
2. **Importing Necessary Libraries**
   - Start your Python script or notebook by importing MNE and other necessary libraries such as NumPy for numerical operations and Matplotlib for plotting. For example: `import mne`, `import numpy as np`, `import matplotlib.pyplot as plt`.
3. **Loading Sample Data**
   - MNE-Python provides sample datasets which are ideal for learning and testing out the library's capabilities.

Setup and Data Handling in MNE-Python

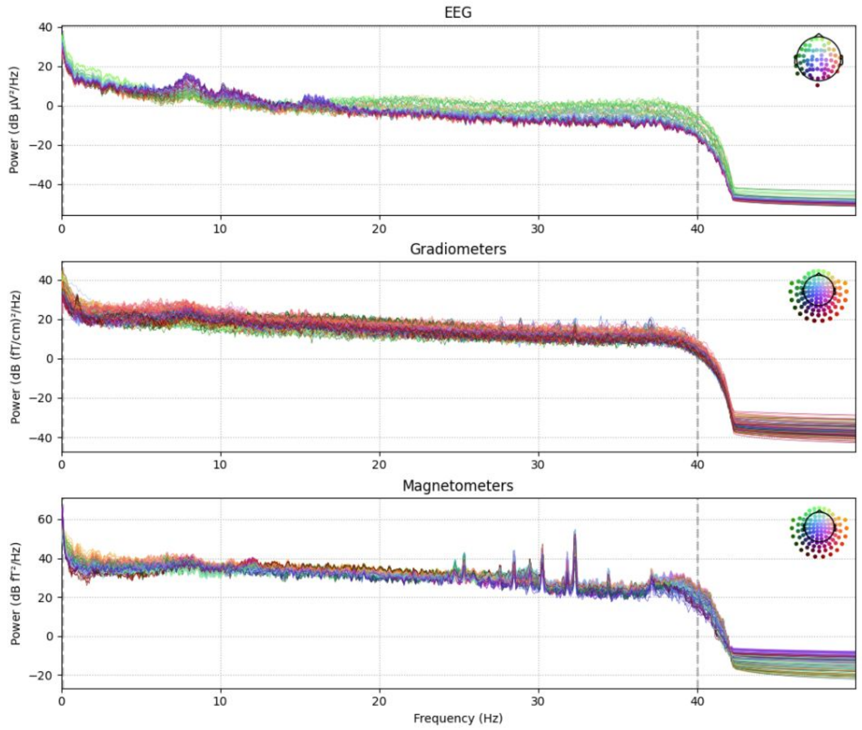


```
<Raw | sample_audvis_filt-0-40_raw.fif, 376 x 41700 (277.7 s), ~3.2 MB, data not loaded>
<Info | 14 non-empty values
 bads: 2 items (MEG 2443, EEG 053)
 ch_names: MEG 0113, MEG 0112, MEG 0111, MEG 0122, MEG 0123, MEG 0121, MEG ...
 chs: 204 Gradiometers, 102 Magnetometers, 9 Stimulus, 60 EEG, 1 EOG
 custom_ref_applied: False
 dev_head_t: MEG device -> head transform
 dig: 146 items (3 Cardinal, 4 HPI, 61 EEG, 78 Extra)
 highpass: 0.1 Hz
 hpi_meas: 1 item (list)
 hpi_results: 1 item (list)
 lowpass: 40.0 Hz
 meas_date: 2002-12-03 19:01:10 UTC
 meas_id: 4 items (dict)
 nchan: 376
 projs: PCA-v1: off, PCA-v2: off, PCA-v3: off, Average EEG reference: off
 sfreq: 150.2 Hz
```

```
Left Auditory
Right Auditory
Left visual
Right visual
```

Loading the Sample Dataset

1. **Loading Data**
   - Sample dataset provided by MNE.
2. **Data Inspection**
   - Inspect it to understand its structure and content : print(raw.info).
   - Number of channels, sampling frequency, and data acquisition duration.
3. **Visualizing Raw Data**
   - Visual inspection of raw data : raw.plot().
   - This interactive plot allows you to scroll through different segments of the data, mark bad channels, and inspect events.



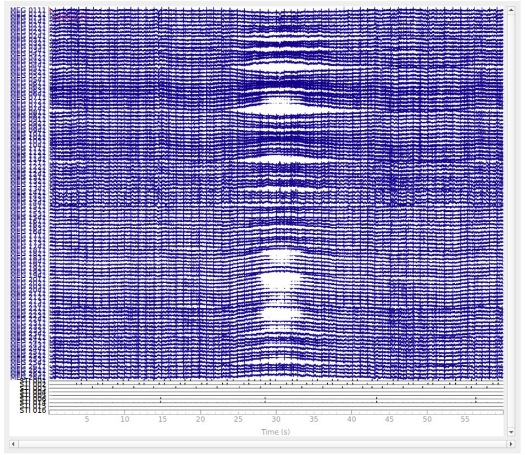
Preprocessing and Inspecting Raw Data

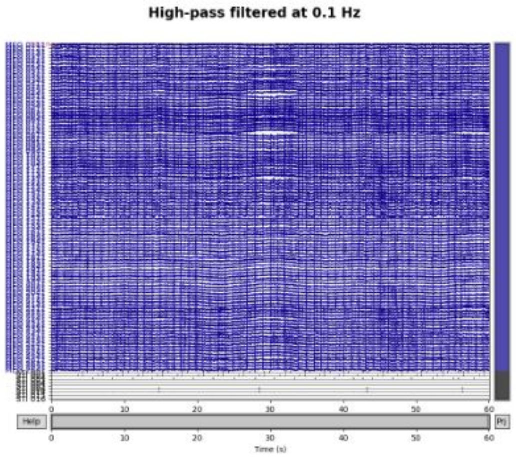
1. **Preprocessing Overview**
   - Preprocessing is crucial for ensuring the quality and reliability of EEG/MEG data analysis.
   - Filtering to remove noise and artifacts, and identifying bad channels that may distort analysis.
2. **Filtering Data**
   - You can apply a band-pass filter to retain a certain frequency range.
   - Inspect the data again to ensure that the filter has been applied correctly and effectively.
   - Visualize the filtered data using to view the power spectral density. This helps in identifying any remaining line noise or other artifacts.
3. **Marking Bad Channels**
   - Identifying and marking bad channels is an essential step in data preprocessing. Bad channels can be channels with excessive noise, flat signals, or abnormal readings.
   - Mark bad channels manually by reviewing the data plot or automatically based on statistical criteria.

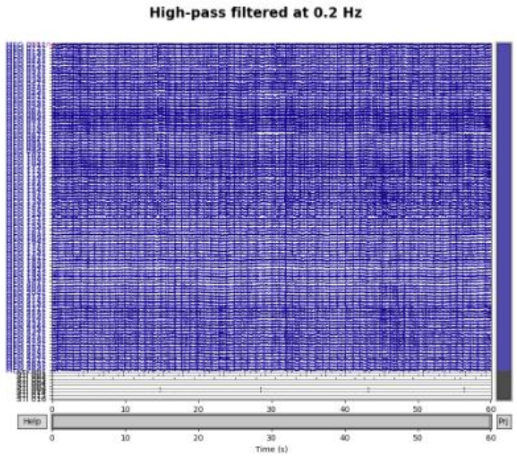
Preprocessing and Inspecting Raw Data

Filtering Data

Original data

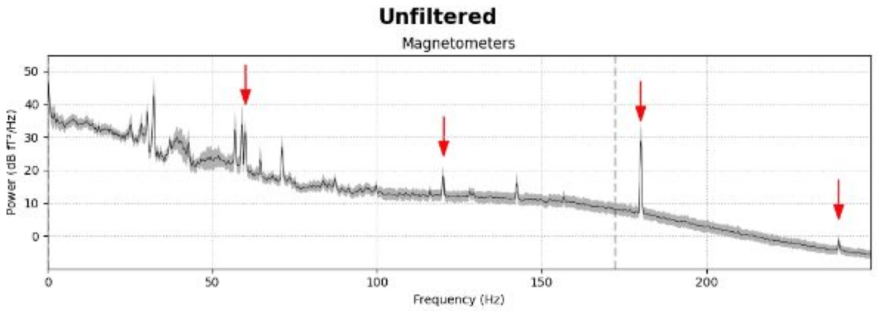


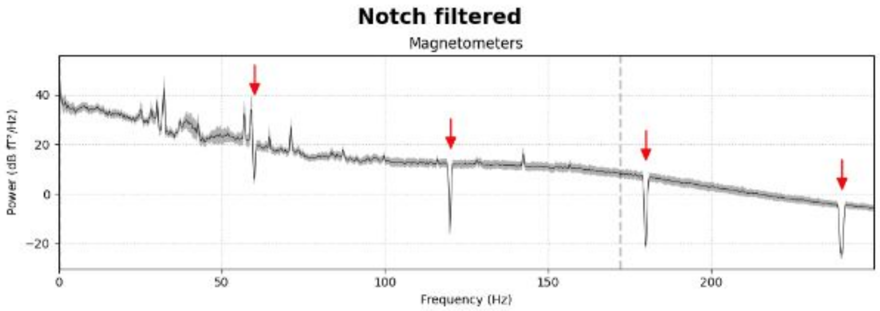




Preprocessing and Inspecting Raw Data

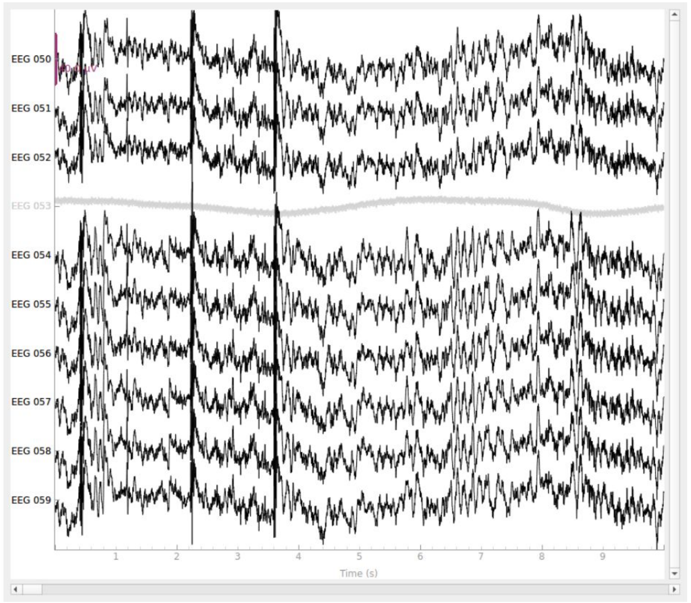
Filtering Data



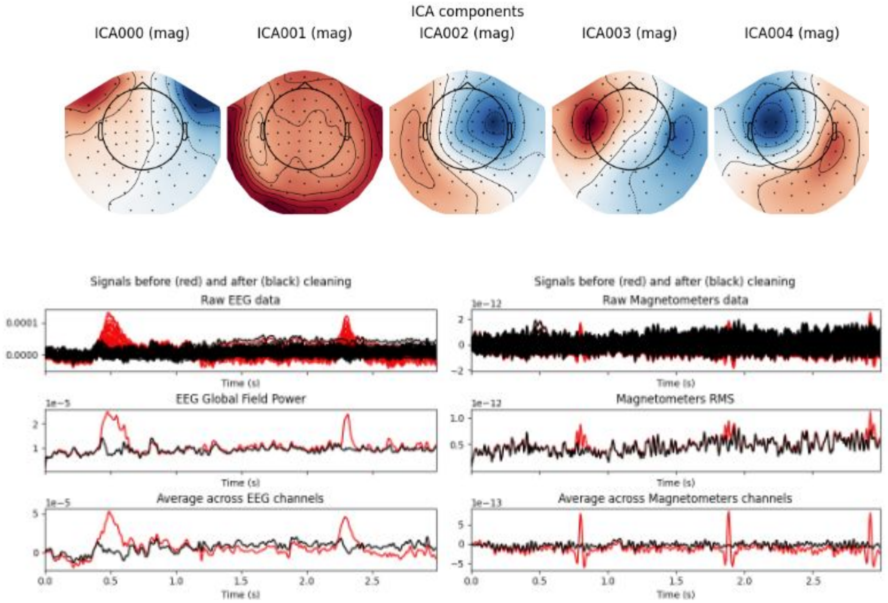


Preprocessing and Inspecting Raw Data

Marking Bad Channels



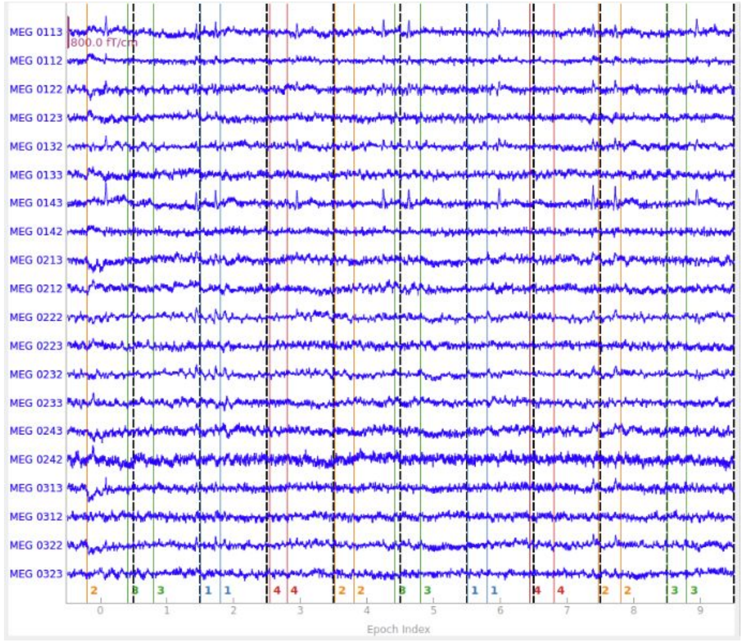
Correcting with ICA

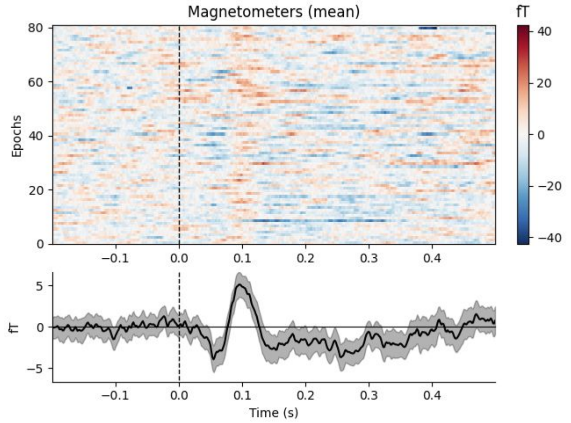


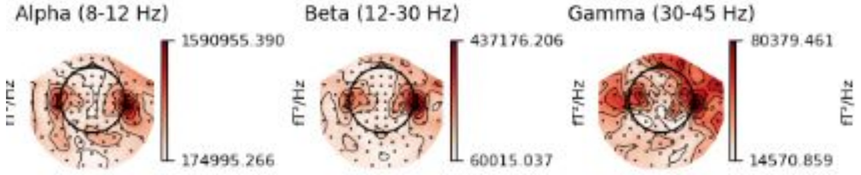
Defining and Extracting Epochs

1. **Introduction to Epochs**
    - Epochs are segments of data extracted around specific events, used for analyzing event-related activities.
    - Time-locked to external stimuli or internal events, such as sensory stimuli or motor actions.
2. **Extracting Events**
    - Identify the events around which to create epochs with triggers: `events, event_id = mne.events_from_annotations(raw)`
3. **Creating Epochs**
    - Pre-event baseline and a post-event time period: `epochs = mne.Epochs(raw, events, event_id, tmin=-0.2, tmax=0.5, baseline=(None, 0), preload=True)`
4. **Inspecting Epochs**
    - `epochs.plot()`, provides an interactive plot to scroll through each epoch and mark bad epochs.

Defining and Extracting Epochs



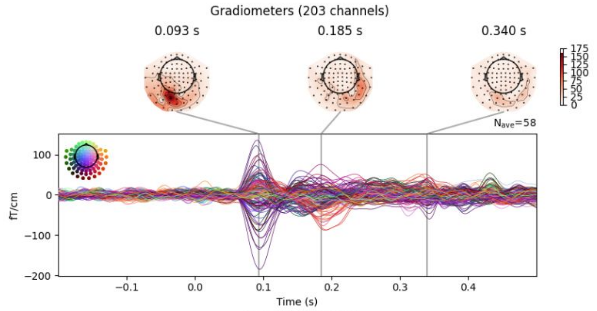


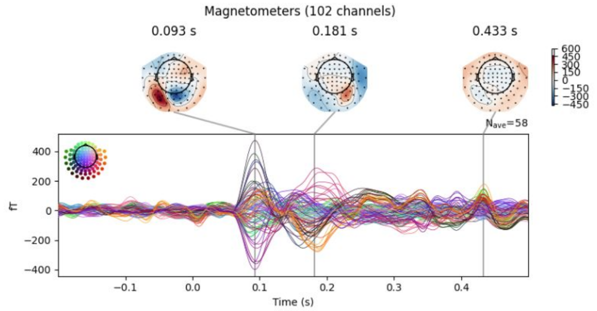


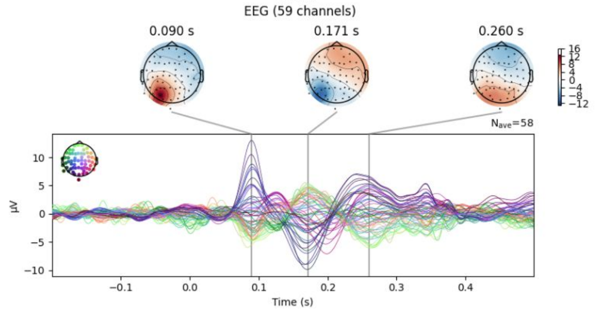
Averaging Epochs and Visualizing Evoked Responses

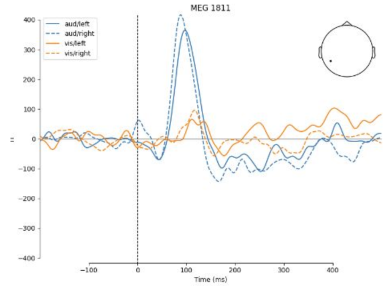
1. **Understanding Evoked Responses**
   - Evoked responses/potentials are averaged signals time-locked to external stimuli or events.
   - Highlights the brain's direct response to specific stimuli.
   - Reduces noise that is not time-locked to the event.
2. **Averaging Epochs**
   - To average the epochs and create an evoked response: `evoked = epochs.average().`
   - Calculates the mean signal across all epochs for each channel.
3. **Visualizing Evoked Responses**
   - `evoked.plot()` to view the evoked response.
   - Peaks corresponding to typical sensory or cognitive components, such as the P300 or N170.

Averaging Epochs and Visualizing Evoked Responses









Section 3

Multivariate Statistics (Decoding/MVPA) - ***script 2***

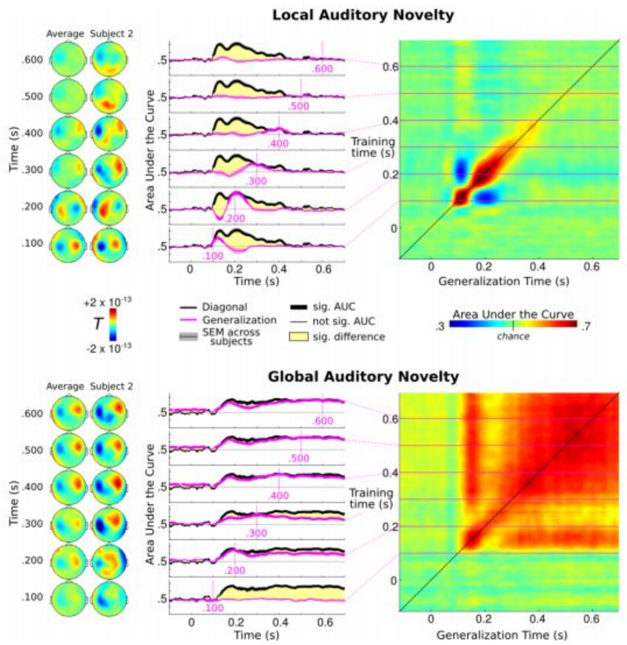
Introduction to Multivariate Analysis

Classification with Linear Discriminant Analysis

Time-resolved Analysis

Introduction to Multivariate Analysis

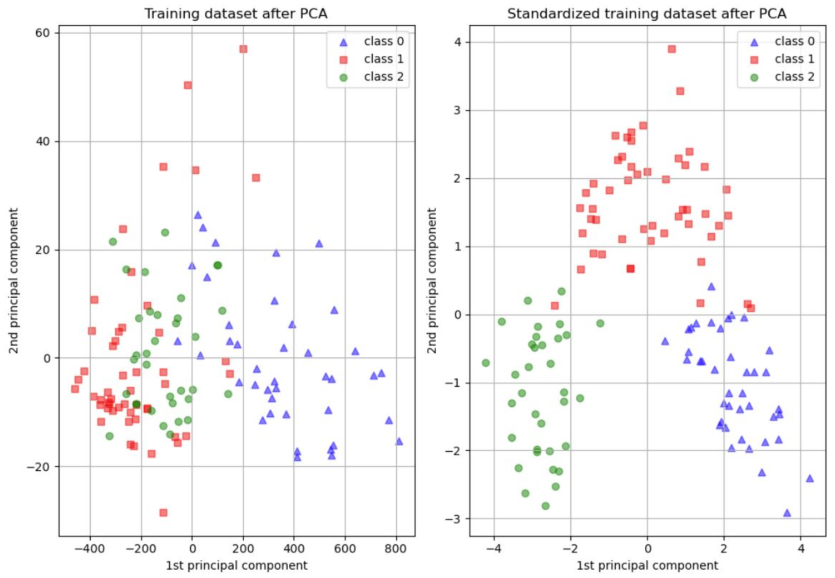
1. **Overview of Multivariate Analysis (Decoding/MVPA)**
   - Statistical techniques to predict or classify data points based on multiple variables.
   - Predicting cognitive states or experimental conditions from complex brain signals.
   - Capture subtle patterns across multiple channels and time points (multivariate).
   - Powerful for analyzing time-resolved neural signals.
   - Insights about the spatial and temporal dynamics of brain function.
2. **Integration with MNE-Python of Scikit-Learn**
   - Scikit-Learn, a machine learning library in Python, to do classification, regression, and clustering algorithms.



King et al., 2014, Plos One

Classification with Linear Discriminant Analysis

1. **Feature Selection**
   - Features can include specific time points, frequency bands that are relevant to the cognitive tasks or states being analyzed.
2. **Data Normalization**
   - Normalize or standardize to ensure that all features contribute equally to the analysis.
   - StandardScaler which removes the mean and scales the data to unit variance.
3. **Setting Up Data Matrices**
   - Organize your data into feature matrices (X) and labels (y). X might consist of the signal amplitudes or power spectral densities at different channels and time points, while y would be the condition labels or participant responses: X = epochs.get_data(); y = epochs.events[:, -1]

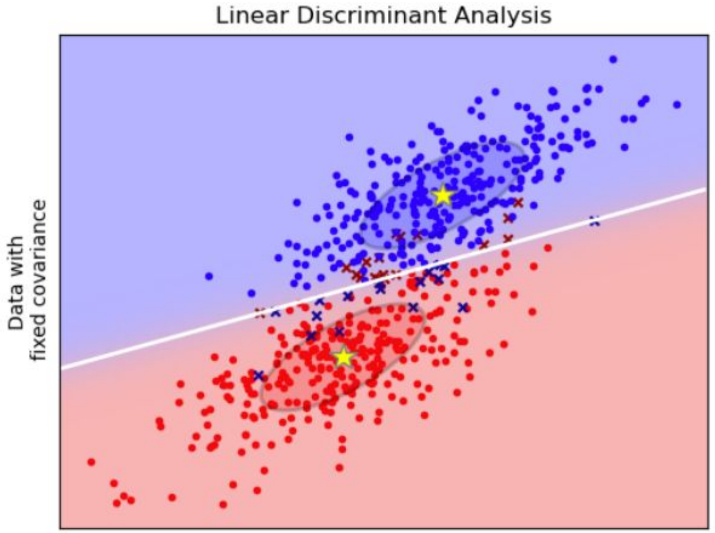


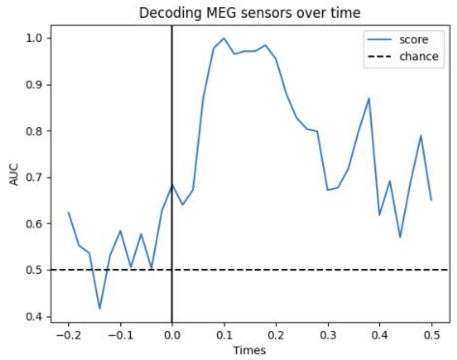
Out:
```
Prediction accuracy for the normal test dataset with PCA
81.48%


Prediction accuracy for the standardized test dataset with PCA
98.15%
```

Classification with Linear Discriminant Analysis

1. **Cross-Validation**
   - To ensure that our model generalizes well to new data.
   - Cross-validation techniques such as KFold or StratifiedKFold.
   - We can create multiple **training and testing splits** to validate the stability and accuracy.
   - from sklearn.model_selection import ShuffleSplit; cv = ShuffleSplit(10, test_size=0.2, random_state=42); scores = cross_val_score(clf, X, y, cv=cv).
2. **Evaluating Classifier Performance**
   - Metrics such as accuracy, precision, recall, or the area under the ROC curve.





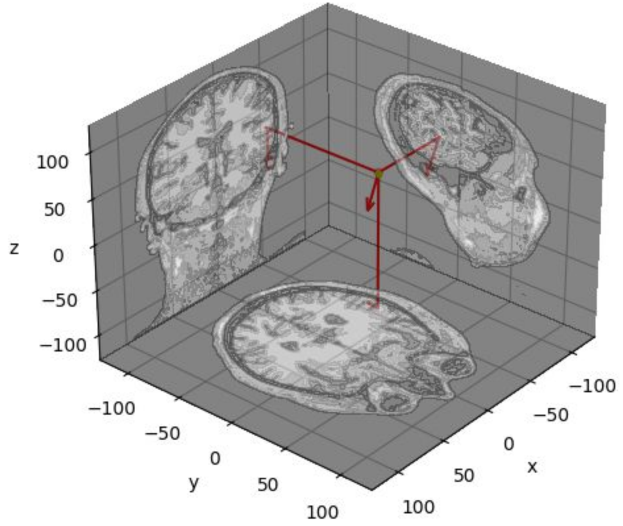
Time-resolved Analysis

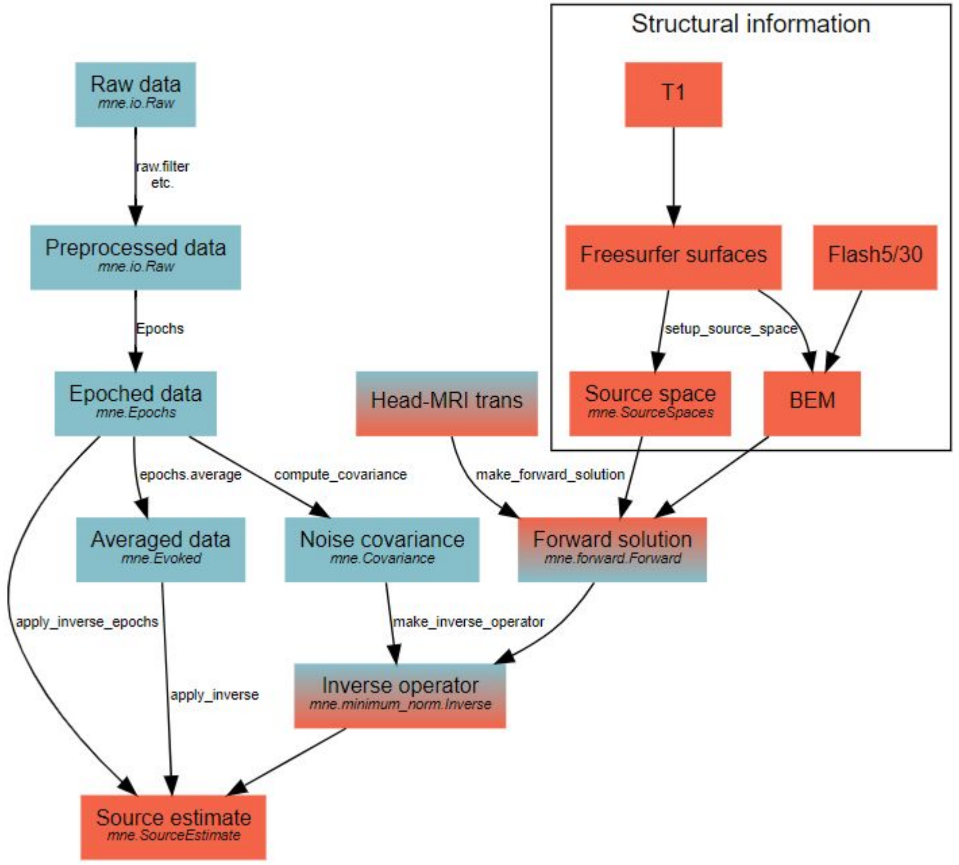
1. **Time-resolved Decoding**
   - Time-resolved decoding involves analyzing how the pattern of brain activity changes over time, providing insights into the dynamics of neural processing.
2. **Implementing Time-resolved Decoding**
   - LDA to perform decoding at each time point across the trial period.
   - from mne.decoding import SlidingEstimator; clf = make_pipeline(StandardScaler(), LinearDiscriminantAnalysis()); time_decod = SlidingEstimator(clf); scores = cross_val_multiscore(time_decod, X, y, cv=5, n_jobs=1); mean_scores = np.mean(scores, axis=0).
3. **Interpreting Results**
   - Understanding which time points show significantly above-chance classification, suggesting key moments of neural differentiation.

Decoding in the Source Space

1. **Setting Up for Source Reconstruction**
   - Source reconstruction involves mapping the observed EEG/MEG data back to the spatial locations in the brain using an inverse operator.
   - inverse_operator = read_inverse_operator(fname_inv)
   - MNE's minimum norm estimate (MNE) to perform source reconstruction from epochs.
2. **Applying the Inverse Model**
   - We apply the inverse model to our epochs data: stcs = apply_inverse_epochs(epochs_meg, inverse_operator, lambda2=1.0 / snr**2, method='dSPM', pick_ori='normal').
3. **Decoding from Source Space**
   - We can visualize the source estimates to see the brain active during the tasks: stc.plot()
   - Decoding cognitive states from the reconstructed source data by transforming the source estimates into feature vectors that can be used by machine learning models: X = np.array([stc.data for stc in stcs]); y = epochs.events[:, 2]

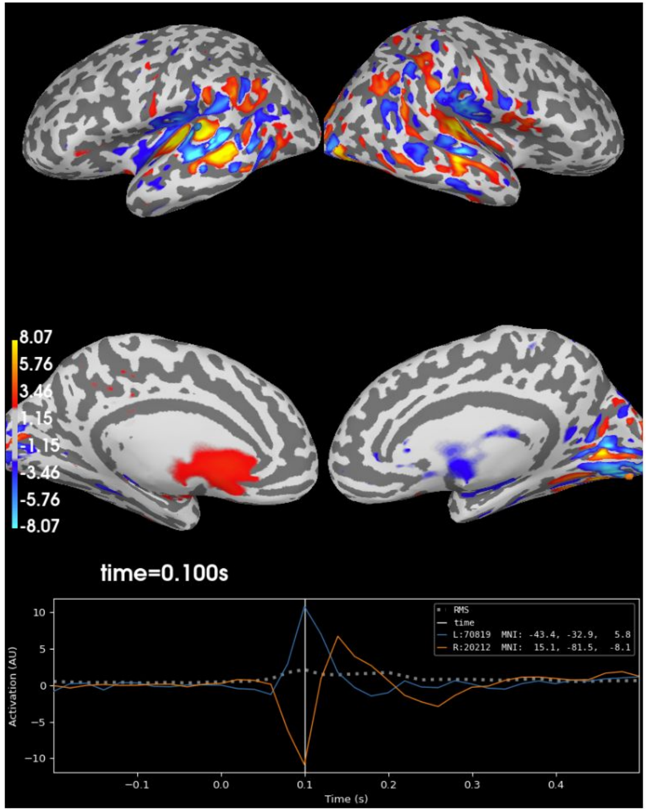
Decoding in the Source Space

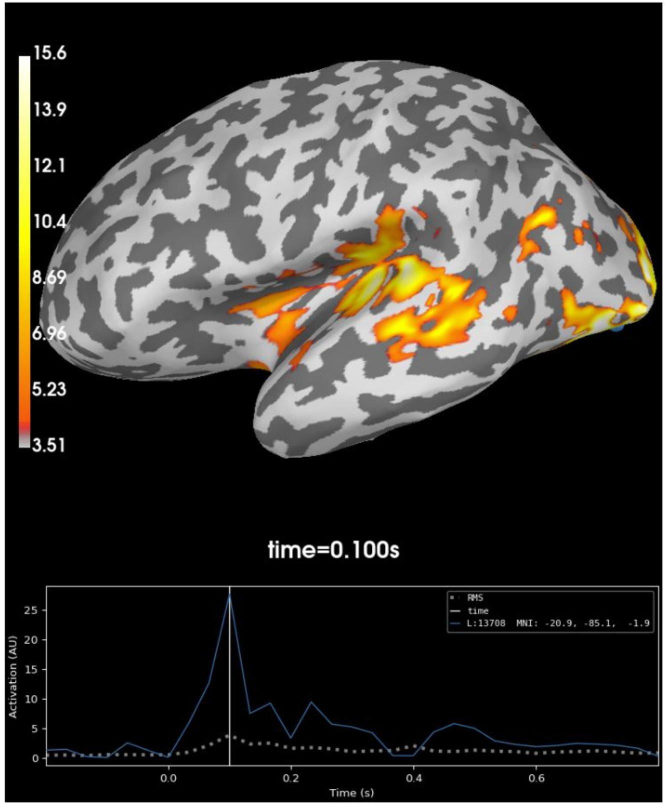




Workflow of the MNE software #

Decoding in the Source Space





Section 4

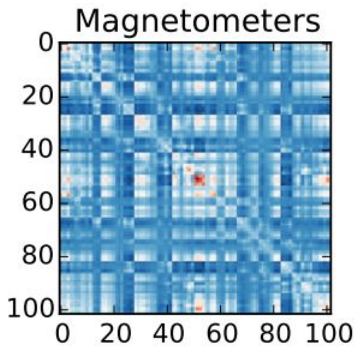
Motor Imagery Decoding with CSP - ***script 3***

Introduction to Motor Imagery Decoding

Data Setup and CSP Application

Integrating CSP with LDA

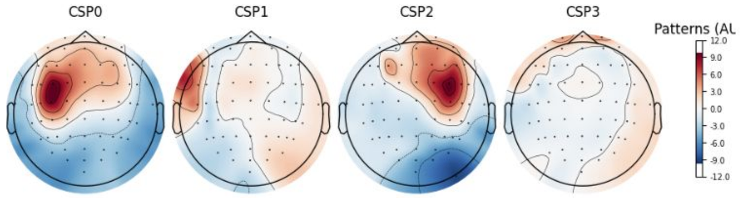
Decoding Motor Imagery with CSP



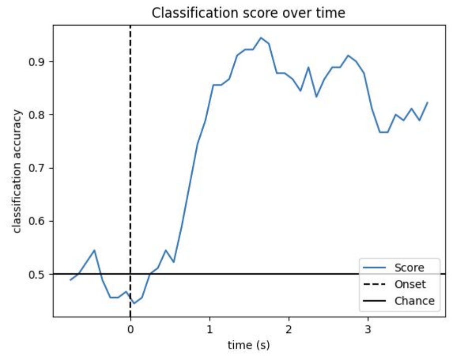
1. **Understanding Motor Imagery Decoding**
    - Motor imagery involves imagining the movement of different parts of the body without actual movement.
    - Map the neural substrates involved in motor planning and execution.
2. **Role of CSP in Motor Imagery**
    - Common Spatial Pattern (CSP) is a statistical technique used to highlight the signal components that have maximal differences between for example two types of motor imagery (e.g., imagining moving hands vs. feet).
    - CSP works by filtering the signal in a way that maximizes the variance for one condition while minimizing it for the other, effectively enhancing the signal-to-noise ratio for classification tasks.
    - Often used in combination with machine learning classifiers.

Applying CSP to EEG Data

1. **Extracting Motor Imagery Events**
   - Band-pass filtering relevant to motor tasks, typically within the 7-30 Hz range.
   - Identify and extract events, e.g., imagining moving hands vs. feet.
2. **Applying CSP**
   - Create epochs focused around these events, with sufficient time window to capture relevant activity.
   - Apply the CSP algorithm to these epochs to derive spatial filters that maximize the variance difference between the two conditions (hands vs. feet): csp = CSP(n_components=4, reg=None, log=True); csp.fit_transform(epochs.get_data(), labels).
3. **Visualizing CSP Filters and Patterns**
   - Visualize the spatial filters and patterns derived from CSP.
   - Filters project onto the scalp, highlighting areas important for distinguishing between the imagined movements: csp.plot_patterns(epochs.info).



Integrating CSP with LDA



1. **Apply LDA**
   - Set up a classifier (LDA) that can use these features to distinguish between moving hands vs. feet.
   - clf = Pipeline([('CSP', CSP(n_components=4)), ('LDA', LinearDiscriminantAnalysis())]).
   - Setting Up Cross-Validation: use the ShuffleSplit method for creating training and testing splits.
   - cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=42).
2. **Evaluating Classifier Performance**
   - Assess the performance of the classifier across different folds of data using the cross-validation setup to assess stability and reliability of the classifier under different subsets of data
   - Calculate and plot cross-validation scores.
   - scores = cross_val_score(clf, X, y, cv=cv)

THANK YOU!

Repository :
https://github.com/thecocolab/mne_meeg_ml_main

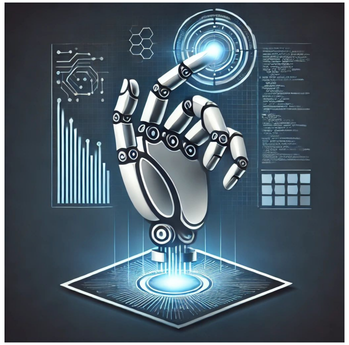
MNE :
https://mne.tools/stable/documentation/index.html

Instructors : Annalisa Pascarella and Vanessa Hadid

MAIN EDUCATIONAL 2024 - Oct 25

Questions?

Section 5 : Hands-On



1. **Practical Exercises**
   - Apply the preprocessing and epoch extraction to the downloaded dataset sample.
   - Identify evoked responses related to different stimuli.
   - Compare classifiers using the CSP-extracted features to decode motor imagery tasks.
   - Explore time-resolved decoding on a dataset of your choice to identify when significant changes in brain activity occur.
2. **Ongoing Learning**
   - We encourage you all to explore the capabilities of MNE-Python and its extensive documentation and tutorials available online.
   - Join online forums and communities related to EEG/MEG research to share your findings, ask questions, and stay updated with the latest developments in the field.