**from** scipy.io **import** loadmat

**import** numpy **as** np

**import** mne

**import** matplotlib.pyplot **as** plt

*# 1. Load the .mat file*

data **=** loadmat('sampleEEGdata.mat')

EEG **=** data['EEG']

eeg\_data **=** EEG['data'][0, 0] *#eeg\_data.shape=(64, 640, 99), [channels:64, time points:640, trials:99]*

times **=** EEG['times'][0, 0][0] *#times.shape= (640,)*

print('EEG.shape:', EEG**.**shape) *#EEG.shape: (1, 1)*

print('eeg\_data.shape:', eeg\_data**.**shape) *#EEG.shape: (1, 1)*

print('times.shape:', times**.**shape) *#EEG.shape: (1, 1)*

microvolt **=** "µV"

*######################################################################################################################################################*

*# a.1. Extract Epochs from 0 to 800 ms*

*# Given a sampling rate of 256 Hz, the time interval between two consecutive samples is 1/256 = 3.90625 ms. (I consider 4ms for each sample point)*

*# Thus, 0 ms corresponds to the sample point at index 256,*

*# and 800 ms corresponds to the sample point at index (800 / 3.90625) = 256+204.8=460.8, which can be rounded to 461.*

eeg\_data\_epoch**=** eeg\_data[:, 256:462, :]*# eeg\_data\_epoch.shape: (64, 206, 99), 256:462:*

*#This indexes elements from the 256th to the 461st along the second dimension(since Python indexing is 0-based and the stop value in a slice is exclusive).*

eeg\_data\_epoch\_20**=** eeg\_data[:, 251:467, :] *# eeg\_data\_epoch\_20.shape:(64, 216, 99), adding the sample point (-20,0) and (800,820) to my epoch.*

*# print('eeg\_data\_epoch.shape:', eeg\_data\_epoch.shape, 'eeg\_data\_epoch\_20.shape', eeg\_data\_epoch\_20.shape )*

*# Compute the ERP at each electrode by averaging across all trials*

ERP **=** np**.**mean(eeg\_data\_epoch, axis**=**2) *# shape: (64, 206)*

ERP\_20 **=** np**.**mean(eeg\_data\_epoch\_20, axis**=**2) *# shape: (64, 216)*

*################################################################*

*# Read the file content*

**with** open('eloc64C2.txt', 'r') **as** f:

lines **=** f**.**readlines()

*# Extract channel names, theta, and radius*

channel\_names **=** [line**.**split()[3]**.**replace('.', '') **for** line **in** lines[0:]] *# Skip the header*

theta **=** np**.**array([float(line**.**split()[1])**-**90 **for** line **in** lines[0:]]) **\*** np**.**pi **/** 180.0 *# Convert to radians*

radius **=** np**.**array([float(line**.**split()[2]) **for** line **in** lines[0:]])

*# Convert polar to Cartesian*

x **=** radius **\*** np**.**cos(theta)**/**5

y **=** **-**radius **\*** np**.**sin(theta)**/**5

z **=** np**.**zeros\_like(x) *# default z-coordinate for all channels*

*# Create a montage*

ch\_pos **=** dict(zip(channel\_names, zip(x, y, z)))

montage **=** mne**.**channels**.**make\_dig\_montage(ch\_pos, coord\_frame**=**'head')

info **=** mne**.**create\_info(ch\_names**=**channel\_names, sfreq**=**256, ch\_types**=**'eeg')

*# Apply the montage to the info*

info**.**set\_montage(montage)

*##########################*

*# Create an evoked object*

*# The evoked object is an instance of the EvokedArray class in MNE, which is useful for visualization and further processing in the MNE ecosystem.*

*#ERP: It is a 2D numpy array that contains the event-related potential values. This is a simple numeric representation with dimensions corresponding to channels and time points.*

*#evoked: It is an object of type EvokedArray that wraps around the ERP numpy array. It integrates the numeric ERP data with metadata (info)*

*#and provides various methods and attributes for easier visualization and analysis of the data in the context of the MNE-Python library.*

evoked **=** mne**.**EvokedArray(ERP, info)

evoked\_20 **=** mne**.**EvokedArray(ERP\_20, info)

*###############################################*

*# a.2. Plot the topographical map*

*# The evoked.plot\_topomap() function from the MNE-Python library is used to plot topographical maps of EEG data,*

*# giving a spatial representation of voltage (or other measures) across the scalp at specific points in time.*

evoked**.**plot\_topomap(times**=**np**.**linspace(0, .8, 9), nrows**=**3, ncols**=**3, size**=**1, cmap**=**'jet', time\_unit**=**'s', show**=False**)

plt**.**suptitle("Time-Averaged Across All Trials \n (Without averaging from 20 ms before to 20 ms after)", y**=**1.05)

plt**.**show()

*##########*

*# Plot the sensor locations*

*# When you're plotting EEG channels with plot\_sensors, MNE sometimes creates two plots: one for a top-down view and another for a side view,*

*# especially when 3D channel locations (with both horizontal and vertical coordinates) are present.*

*# You can suppress the side view by setting the proj argument to True:*

evoked**.**plot\_sensors(kind**=**'topomap', show\_names**=True**, ch\_type**=**'eeg')

*#######################################################################*

*# a. Time-Averaged Topographical Maps for Enhanced Signal-to-Noise Ratio*

plt**.**close('all')

center\_times **=** np**.**linspace(0, .8, 9) *# 0ms, 100ms, ..., 800ms*

fig, axs **=** plt**.**subplots(3, 3, figsize**=**(10,8))

fig**.**suptitle("Time-Averaged Topographical Maps for Enhanced Signal-to-Noise Ratio", fontsize**=**12, y**=**1.05)

AVE\_ERP**=**[]

**for** idx, center\_time **in** enumerate(center\_times):

start\_time **=** center\_time **-** 0.020

end\_time **=** center\_time **+** 0.020

avg\_data **=** evoked\_20**.**copy()**.**crop(tmin**=**start\_time, tmax**=**end\_time)**.**data**.**mean(axis**=**1)

AVE\_ERP**.**append(avg\_data)

ax **=** axs[idx **//** 3, idx **%** 3]

im, \_ **=** mne**.**viz**.**plot\_topomap(avg\_data, evoked\_20**.**info, cmap**=**'jet', axes**=**ax, show**=False**)

fig**.**colorbar(im, ax**=**ax, format**=**'%3.1f', orientation**=**'vertical', label**=**'Voltage (µV)')

ax**.**set\_title(f"Center: {center\_time**\***1000:.0f} ms\nMean: ({start\_time**\***1000:.0f} ms to {end\_time**\***1000:.0f} ms)")

plt**.**tight\_layout()

plt**.**show()

AVE\_ERP\_np**=**np**.**array(AVE\_ERP)

*# print('AVE\_ERP\_np.shape', AVE\_ERP\_np.shape) #(9,64)*

*########################################################*

*# Making the color limit for all the topoplots the same*

plt**.**close('all')

center\_times **=** np**.**linspace(0, .8, 9) *# 0ms, 100ms, ..., 800ms*

fig, axs **=** plt**.**subplots(3, 3, figsize**=**(10, 8))

fig**.**suptitle("Time-Averaged Topographical Maps for Enhanced Signal-to-Noise Ratio\n (the color limit for all the topoplots is the same)", fontsize**=**12, y**=**1.05)

global\_vmin, global\_vmax **=** AVE\_ERP\_np**.**min(), AVE\_ERP\_np**.**max()

**for** idx, center\_time **in** enumerate(center\_times):

start\_time **=** center\_time **-** 0.020

end\_time **=** center\_time **+** 0.020

ax **=** axs[idx **//** 3, idx **%** 3]

im, \_ **=** mne**.**viz**.**plot\_topomap(AVE\_ERP\_np[idx, :], evoked\_20**.**info, cmap**=**'jet', axes**=**ax, show**=False**)

*# Adjust the color limits to reflect the global max and min*

im**.**set\_clim(global\_vmin, global\_vmax)

fig**.**colorbar(im, ax**=**ax, format**=**'%3.1f', orientation**=**'vertical', label**=**'Voltage (µV)')

ax**.**set\_title(f"Center: {center\_time**\***1000:.0f} ms\nMean: ({start\_time**\***1000:.0f} ms to {end\_time**\***1000:.0f} ms)")

plt**.**tight\_layout()

plt**.**show()

*# print('AVE\_ERP\_np.shape', AVE\_ERP\_np.shape) # (9, 64)*

EEG.shape: (1, 1)

eeg\_data.shape: (64, 640, 99)

times.shape: (640,)

A chart of a graph

Description automatically generated with medium confidence

A diagram of a circle with many dots

Description automatically generated

C:\Users\tnlab\AppData\Local\Temp\ipykernel\_16184\3022459550.py:77: RuntimeWarning: tmin is not in time interval. tmin is set to <class 'mne.evoked.EvokedArray'>.tmin (0 s)

avg\_data = evoked\_20.copy().crop(tmin=start\_time, tmax=end\_time).data.mean(axis=1)

A group of maps of different colors

Description automatically generated

A group of colored circles with numbers and lines

Description automatically generated with medium confidence

In [291]:

*# b.*

times **=** evoked**.**times

*# print('times:', times, 'times.shape', times.shape) # [0. 0.00390625, ..., 0.80078125], times.shape (206,)*

max\_values **=** []

max\_times **=** []

**for** i **in** range(ERP**.**shape[0]):

max\_val **=** np**.**max(ERP[i, :])

max\_idx **=** np**.**argmax(ERP[i, :])

max\_values**.**append(max\_val)

max\_times**.**append(times[max\_idx])

max\_values **=** np**.**array(max\_values)

max\_times **=** np**.**array(max\_times) **\*** 1000 *# Convert to milliseconds*

*######################################################################*

fig, ax **=** plt**.**subplots(figsize**=**(6,5))

im, \_ **=** mne**.**viz**.**plot\_topomap(max\_times, evoked**.**info, cmap**=**'jet', contours**=**0, axes**=**ax, show**=False**, size**=**3)

*# Show colorbar*

plt**.**colorbar(im, ax**=**ax, label**=**'Peak Time (ms)')

plt**.**title("Peak Time of the ERP")

plt**.**show()

A rainbow colored circle with a scale and a measuring scale

Description automatically generated with medium confidence

b. In the displayed figure, the color scheme indicates the timing of responses, where blue represents the earliest response and red represents the latest. Shades in between denote relative response timings based on their color intensity. Upon examination, electrodes (Cpz, Cp2, Cp1, P1, P10, P9) exhibit the earliest response. This trend is also evident in the upper left section of the scalp. Conversely, electrodes like Iz, Fp2, Af8, and F8 show the latest response. Additionally, below you can find the temporal average visualization for each electrode, presented separately

In [292]:

*############*

fig, axs **=** plt**.**subplots(nrows**=**16, ncols**=**4, figsize**=**(15, 40))

fig**.**tight\_layout(pad**=**2.0) *# Adjust the spacing between subplots*

fig**.**suptitle('Temporal Average Visualisation For Each Electrode', fontsize**=**16, y**=**1.01)

*# Flatten the axis array for easier indexing*

axs **=** axs**.**ravel()

**for** idx **in** range(64): *# Loop through each channel*

*# Plot the ERP data for the current channel over time*

axs[idx]**.**plot(times **\*** 1000, ERP[idx, :])

axs[idx]**.**set\_title(evoked**.**ch\_names[idx]) *# Set title as channel name*

axs[idx]**.**set\_xlabel('Time (ms)')

axs[idx]**.**set\_ylabel('Amplitude (µV)')

plt**.**subplots\_adjust(hspace**=**0.5)

plt**.**show()

A chart of blue lines

Description automatically generated with medium confidence

In [294]:

*# c.*

**with** open('eloc64C2.txt', 'r') **as** f\_l:

lines\_l **=** f\_l**.**readlines()

channel\_names\_l **=** [line\_l**.**split()[3]**.**replace('.', '') **for** line\_l **in** lines\_l[0:]]

theta\_l **=** np**.**array([float(line\_l**.**split()[1]) **for** line\_l **in** lines\_l[0:]]) **\*** np**.**pi **/** 180.0

radius\_l **=** np**.**array([float(line\_l**.**split()[2]) **for** line\_l **in** lines\_l[0:]])

x\_l **=** radius\_l **\*** np**.**cos(theta\_l)

y\_l **=** radius\_l **\*** np**.**sin(theta\_l)

z\_l **=** np**.**zeros\_like(x\_l)

radius\_range **=** [0.18, 0.28]

laplacian\_data **=** np**.**zeros\_like(ERP\_20)

**for** i **in** range(64):

distances **=** np**.**sqrt((x\_l **-** x\_l[i]) **\*\*** 2 **+** (y\_l **-** y\_l[i]) **\*\*** 2)

surrounding\_indices **=** np**.**where((distances **>** radius\_range[0]) **&** (distances **<** radius\_range[1]))[0]

**if** len(surrounding\_indices) **>** 0:

weights **=** 1 **/** distances[surrounding\_indices]

weights **=** weights **/** np**.**sum(weights) *# Normalize weights*

**else**:

weights **=** np**.**array([])

weights **=** weights[:, np**.**newaxis]

**if** len(weights) **>** 0:

laplacian\_data[i, :] **=** ERP\_20[i, :] **-** np**.**sum(weights **\*** ERP\_20[surrounding\_indices, :], axis**=**0)

**else**:

laplacian\_data[i, :] **=** ERP\_20[i, :]

laplacian\_raw **=** evoked\_20**.**copy()

laplacian\_raw**.**\_data **=** laplacian\_data

*########################################*

plt**.**close('all')

center\_times **=** np**.**linspace(0, .8, 9)

fig, axs **=** plt**.**subplots(3, 3, figsize**=**(10,8))

fig**.**suptitle("Time-Averaged Topographical Maps for Enhanced Signal-to-Noise Ratio", fontsize**=**12, y**=**1.03)

AVE\_ERP\_l**=**[]

**for** idx, center\_time **in** enumerate(center\_times):

start\_time **=** center\_time **-** 0.020

end\_time **=** center\_time **+** 0.020

avg\_data\_l **=** laplacian\_raw**.**copy()**.**crop(tmin**=**start\_time, tmax**=**end\_time)**.**data**.**mean(axis**=**1)

AVE\_ERP\_l**.**append(avg\_data)

ax **=** axs[idx **//** 3, idx **%** 3]

im, \_ **=** mne**.**viz**.**plot\_topomap(avg\_data\_l, laplacian\_raw**.**info, cmap**=**'jet', axes**=**ax, show**=False**)

fig**.**colorbar(im, ax**=**ax, format**=**'%3.1f', orientation**=**'vertical', label**=**'Voltage (µV)')

ax**.**set\_title(f"Center: {center\_time**\***1000:.0f} ms\nMean: ({start\_time**\***1000:.0f} ms to {end\_time**\***1000:.0f} ms)")

plt**.**tight\_layout()

plt**.**show()

AVE\_ERP\_np\_l**=**np**.**array(AVE\_ERP)

*###################################################*

plt**.**close('all')

center\_times **=** np**.**linspace(0, .8, 9)

fig, axs **=** plt**.**subplots(3, 3, figsize**=**(10, 8))

fig**.**suptitle("Time-Averaged Topographical Maps for Enhanced Signal-to-Noise Ratio\n (the color limit for all the topoplots is the same)", fontsize**=**12, y**=**1.03)

global\_vmin, global\_vmax **=** AVE\_ERP\_np\_l**.**min(), AVE\_ERP\_np\_l**.**max()

**for** idx, center\_time **in** enumerate(center\_times):

start\_time **=** center\_time **-** 0.020

end\_time **=** center\_time **+** 0.020

ax **=** axs[idx **//** 3, idx **%** 3]

im, \_ **=** mne**.**viz**.**plot\_topomap(AVE\_ERP\_np\_l[idx, :], laplacian\_raw**.**info, cmap**=**'jet', axes**=**ax, show**=False**)

im**.**set\_clim(global\_vmin, global\_vmax)

fig**.**colorbar(im, ax**=**ax, format**=**'%3.1f', orientation**=**'vertical', label**=**'Voltage (µV)')

ax**.**set\_title(f"Center: {center\_time**\***1000:.0f} ms\nMean: ({start\_time**\***1000:.0f} ms to {end\_time**\***1000:.0f} ms)")

plt**.**tight\_layout()

plt**.**show()

C:\Users\tnlab\AppData\Local\Temp\ipykernel\_16184\2574989804.py:37: RuntimeWarning: tmin is not in time interval. tmin is set to <class 'mne.evoked.EvokedArray'>.tmin (0 s)

avg\_data\_l = laplacian\_raw.copy().crop(tmin=start\_time, tmax=end\_time).data.mean(axis=1)

A group of maps of different colors

Description automatically generated

A group of maps of different colors

Description automatically generated

By implementing the large Laplacian filter, we significantly enhance the spatial precision of the EEG data. The Laplacian filter operates on the principle of spatial second derivatives, which emphasizes local changes in the signal. As a result, this technique sharpens the topographic representation of the electrical activity on the scalp, allowing for a more detailed and localized observation. When comparing the results from step (c) with those from step (a), it becomes evident that the Laplacian-filtered data provides a clearer distinction between active and non-active regions. This improved localization can be particularly beneficial in identifying focal brain activities and can aid in the interpretation of the underlying neural sources. Thus, the large Laplacian filter emerges as an essential tool for a more precise spatial analysis of EEG data