**CÓDIGO EMOTIV EPOX X 14 – PYTHON – MNE LIBRARY**

PREPROCESSING, PROCESSING, MAPEO Y ANÁLISIS ESTADÍSTICO

Diagrama

Descripción generada automáticamente

**\*Nota:** Enlace para mejor visualización del esquema (<https://cards.algoreducation.com/app/set/6763e6cceef1352c47ffdab4?sheetId=6763e6cfeef1352c47ffdadb> )

Git Emotiv: <https://github.com/Emotiv/cortex-example> ; <https://emotiv.gitbook.io/cortex-api>

**IMPORTAR DATOS**

<https://sccn.ucsd.edu/wiki/Makoto's_preprocessing_pipeline>

Código ejemplo para importar datos de EMOITV EPOC en formato .EDF:

import mne

path = "/Users/anarubiomorales/Downloads/OA post\_EPOCX\_200734\_2023.11.08T13.51.29+01.00.edf" raw = mne.io.read\_raw\_edf(path)

**\*Nota:** Este código habría que mejorarlo para que importe fácilmente todos los archivos de una carpeta, o bien, todos os archivos que contengan tal/es palabra/s.

Posible solución:

# Lista de archivos que quieres procesar

archivos = [f"P\_OApre\_psd\_bands\_{i}\_emotiv.csv" for i in range(1, 13)]

**PREPROCESSING**

**1º Resampling (raw.resample)**

El casco lo tenemos sampleando a 256 Hz, entonces: 128 Hz

Teoría:

La frecuencia de Nyquist es la mitad de la frecuencia de muestreo y marca la frecuencia máxima que puede representarse sin errores al digitalizar una señal. Si grabas a 256 Hz, la frecuencia de Nyquist es 128 Hz; al hacer resampling a 128 Hz, pasa a ser 64 Hz. Antes de reducir el muestreo, se filtran las frecuencias mayores al nuevo límite (64 Hz) para evitar distorsión (aliasing).

<https://en.wikipedia.org/wiki/Nyquist_frequency>

Código:

raw = raw.resample(125)

**2º Bp filter** *(cuando pasa filtros probablemente sobre eso te haga fft y luego te la deshaga! )*

High pass – 50 Hz

Low pass – 1 Hz

Código:

raw.filter(l\_freq=1, h\_freq=50)

**3º Crear y definir el montaje.**

ch\_names = ['AF3', 'F7', 'F3', 'FC5', 'T7', 'P7', 'O1', 'O2', 'P8', 'T8', 'FC6', 'F4', 'F8', 'AF4']

raw = raw.pick(ch\_names)

montage = mne.channels.make\_standard\_montage('standard\_1020')

raw.info.set\_montage(montage)

raw.info.get\_montage().get\_positions()

raw.info.get\_montage().plot()

pass

**4º Prep (canales)**

Librería PyPREP (<https://github.com/sappelhoff/pyprep>)

Código:

from pyprep.prep\_pipeline import PrepPipeline as pyprep\_PrepPipeline

 prep\_params = {

            "ref\_chs": "eeg",

            "reref\_chs": "eeg",

            "line\_freqs": [np.arange(60, sample.sfreq / 2, 60)],

            "max\_iterations": 8,

        }

prep\_pipeline = pyprep\_PrepPipeline(raw, prep\_params, raw.info.get\_montage())

prep\_pipeline.fit()

print("Bad channels: %s", prep\_pipeline.interpolated\_channels)

print("Bad channels original: %s", prep\_pipeline.noisy\_channels\_original["bad\_all"] )

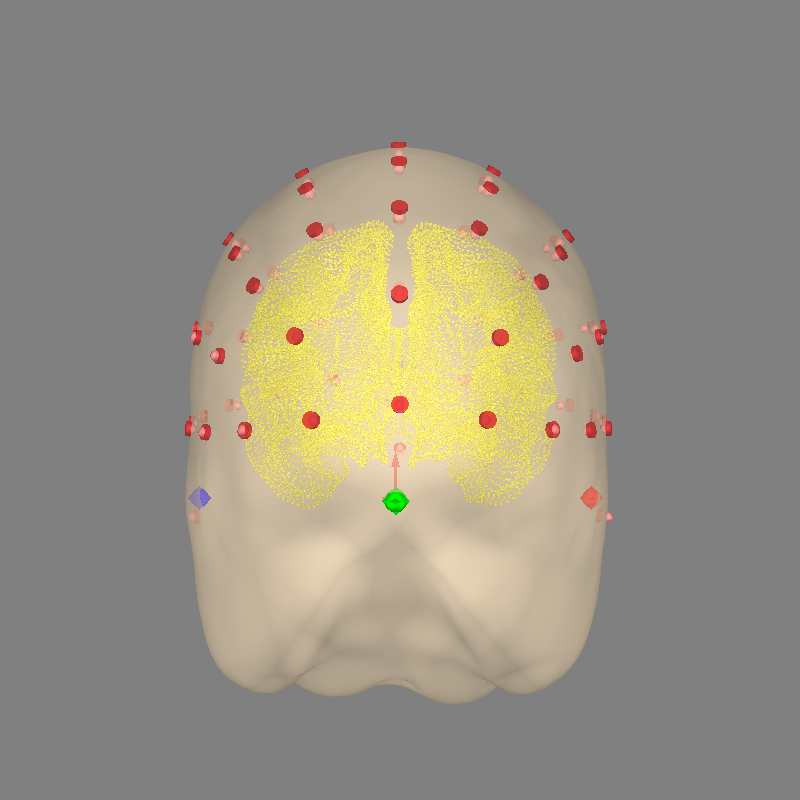
print("Bad channels after interpolation: %s", prep\_pipeline.still\_noisy\_channels)

raw = prep\_pipeline.raw.copy()

**5º Average**

Teoría:

3 puntos de referencia (fiducials references) à Orejas y nariz



**6º ICA**

1. **PCA à ANA**
2. **ICA**

Teoría:

**Un dibujo de un pizarrón blanco

Descripción generada automáticamente con confianza baja**

Pero ponemos components0.9 o 0.99 (porque así de esta forma al no saber cuántos componentes hay (en el ejemplo se ve que hay 3), pues coge el 99 o 90% de los “instrumentos, componentes random del cerebro)

Mne.preprocesingICA à PSD de ICA

4 CASOS :

1. Downsampling (no sale en paralelo)
2. –
3. …
4. ---

Código:

fit\_params = dict(tol=self.tol) # default is 1e-4

cov = mne.compute\_raw\_covariance(sample.raw)

self.ica = mne.preprocessing.ICA(method=self.method,n\_components=self.n\_components, noise\_cov=cov, max\_iter=self.max\_iter, fit\_params=fit\_params, verbose=True)

self.ica.fit(sample.raw)

ica\_srcs = self.ica.get\_sources(sample.raw).get\_data()

# the criteria for identifying bad components includes

# the computation of the z-scores of each component’s

# spectral kurtosis, slope, Hurst exponent and gradient median

power\_spectrums = np.abs(np.fft.fft(ica\_srcs,axis=1))\*\*2

time\_step = 1 / sample.sfreq

frequencies = np.fft.fftfreq(ica\_srcs.shape[1], time\_step)

spec\_kurtosis = scipy.stats.kurtosis(power\_spectrums,axis=1)

hurst\_exponents = get\_hurst\_exponents(ica\_srcs)

mean\_slopes\_pd = calculate\_mean\_slope\_numpy(power\_spectrums, axis=1)

median\_slopes = calculate\_mean\_slope\_numpy(ica\_srcs,median=True, axis=1)

spec\_kurtosis\_zs = scipy.stats.zscore(spec\_kurtosis, axis=None)

spec\_kurtosis\_mask = np.logical\_or(spec\_kurtosis\_zs < self.min\_z, spec\_kurtosis\_zs > self.max\_z)

hurst\_exponents\_zs = scipy.stats.zscore(hurst\_exponents, axis=None)

hurst\_exponents\_mask = np.logical\_or(hurst\_exponents\_zs < self.min\_z, hurst\_exponents\_zs > self.max\_z)

mean\_slopes\_pd\_zs = scipy.stats.zscore(mean\_slopes\_pd, axis=None)

mean\_slopes\_pd\_mask = np.logical\_or(mean\_slopes\_pd\_zs < self.min\_z, mean\_slopes\_pd\_zs > self.max\_z)

median\_slopes\_zs = scipy.stats.zscore(median\_slopes, axis=None)

median\_slopes\_mask = np.logical\_or(median\_slopes\_zs < self.min\_z, median\_slopes\_zs > self.max\_z)

exclude\_mask = np.logical\_or(spec\_kurtosis\_mask, hurst\_exponents\_mask)

exclude\_mask = np.logical\_or(exclude\_mask, mean\_slopes\_pd\_mask)

exclude\_mask = np.logical\_or(exclude\_mask, median\_slopes\_mask)

self.ica.exclude = np.where(exclude\_mask)[0]

print(f"[ICA]: BAD by spectral kurtosis ({spec\_kurtosis\_mask.sum() })ids={np.where(spec\_kurtosis\_mask)[0] }")

print(f"[ICA]: BAD by hurst exponent ({hurst\_exponents\_mask.sum()}) ids={np.where(hurst\_exponents\_mask)[0] }")

print(f"[ICA]: BAD by mean slope of power density ({mean\_slopes\_pd\_mask.sum()}) ids={np.where(mean\_slopes\_pd\_mask)[0] }")

print(f"[ICA]: BAD by median slope of ic ({median\_slopes\_mask.sum()}) ids={np.where(median\_slopes\_mask)[0] }")

print(f"[ICA]: {self.ica.exclude= }")

explained\_variance\_ = self.ica.pca\_explained\_variance\_

explained\_variance\_ratio\_ = explained\_variance\_ / np.sum(explained\_variance\_)

# ica.plot\_components(picks=None,show=True, inst=sample.raw)

# print(f"{ica.unmixing\_matrix\_.shape= } {ica\_srcs.shape= }")

raw = self.ica.apply(raw)

**PROCESING**

**PSD … : Welch’s methods** <https://en.wikipedia.org/wiki/Welch%27s_method>

<https://notebook.community/JoseGuzman/myIPythonNotebooks/SignalProcessing/Welch's%20periodogram>

El método de Welch divide la señal EEG en segmentos superpuestos, calcula la potencia de cada segmento usando la Transformada de Fourier y luego promedia los resultados para reducir el ruido. Esto muestra cómo se distribuye la energía en diferentes frecuencias. En Python, se implementa fácilmente con scipy.signal.welch.

**COLOCACIÓN ELECTRODOS Y MAPEO**

<https://www.emotiv.com/blogs/how-to/understanding-the-10-20-system-of-eeg-electrode-placement>

<https://www.researchgate.net/publication/361688280_Effect_of_time_windows_in_LSTM_networks_for_EEG-based_BCIs/figures?lo=1>

Código:

pip install mne pandas matplotlib

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import mne

# Ruta a tus archivos

carpeta\_datos = "/Users/rosaayusomoreno/Library/CloudStorage/OneDrive-UniversidaddeExtremadura/0. ROSA AYUSO/Proyectos/ESCALADA/Escalada Cognitiva (Fiore)/2.Datos\_Análisis/ResultadosEEG"

carpeta\_salida = carpeta\_datos # Guardar imágenes en la misma carpeta

# Lista de archivos que quieres procesar

archivos = [f"P\_OApre\_psd\_bands\_{i}\_emotiv.csv" for i in range(1, 13)]

# Coordenadas y nombres de los electrodos (EMOTIV EPOC - 14 canales)

electrodos = ['AF3', 'F7', 'F3', 'FC5', 'T7', 'P7', 'O1',

'O2', 'P8', 'T8', 'FC6', 'F4', 'F8', 'AF4']

posiciones\_electrodos = {

'AF3': [-0.2, 0.5, 0.0], 'F7': [-0.5, 0.3, 0.0], 'F3': [-0.2, 0.3, 0.0], 'FC5': [-0.3, 0.1, 0.0],

'T7': [-0.5, -0.1, 0.0], 'P7': [-0.4, -0.3, 0.0], 'O1': [-0.2, -0.5, 0.0], 'O2': [0.2, -0.5, 0.0],

'P8': [0.4, -0.3, 0.0], 'T8': [0.5, -0.1, 0.0], 'FC6': [0.3, 0.1, 0.0], 'F4': [0.2, 0.3, 0.0],

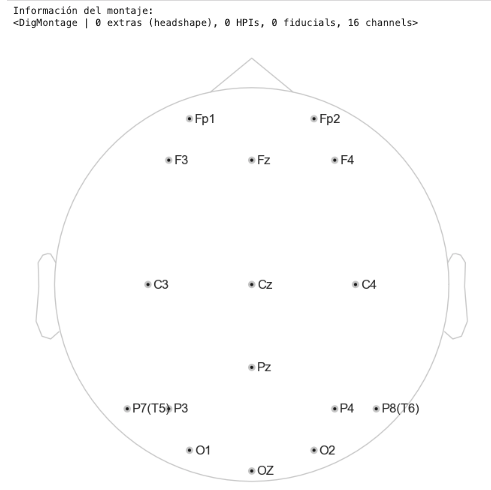
'F8': [0.5, 0.3, 0.0], 'AF4': [0.2, 0.5, 0.0]

}

# Crear el montaje manual

montage = mne.channels.make\_dig\_montage(ch\_pos=posiciones\_electrodos, coord\_frame="head")

**Nota\*:**IMPORTANTE COMPROBAR ESTAS COORDENADAS si son de EMOTIV o no. Creo que quizá es mejor usar las de MNE validadas. ANA CREO QUE CARLOS TE METIO EL CODIGO

Imagen con coordenadas manuales

for i, archivo in enumerate(archivos, start=1):

ruta\_archivo = os.path.join(carpeta\_datos, archivo)

if not os.path.exists(ruta\_archivo):

print(f"No se encontró el archivo: {ruta\_archivo}")

continue

# Leer el archivo CSV

try:

datos\_eeg = pd.read\_csv(ruta\_archivo, index\_col=0) # Usar la primera columna como índice

print(f"Archivo {archivo} leído correctamente. Dimensiones: {datos\_eeg.shape}")

except Exception as e:

print(f"Error al leer {archivo}: {e}")

continue

# Eliminar la última fila (ECG\_artificial)

if 'ECG\_artificial' in datos\_eeg.index:

datos\_eeg = datos\_eeg.drop('ECG\_artificial', axis=0)

print(f"Eliminada la fila 'ECG\_artificial' en {archivo}")

# Verificar si los datos tienen el tamaño correcto

if len(datos\_eeg) != len(electrodos):

print(f"Error en {archivo}: El número de canales no coincide después de eliminar 'ECG\_artificial'.")

continue

# Crear la información del EEG

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

info.set\_montage(montage) # Aplicar el montaje

# Generar mapas de calor para cada banda

bandas = datos\_eeg.columns # Las bandas son las columnas

for j, banda in enumerate(bandas):

valores = datos\_eeg[banda].to\_numpy() # Obtener los valores de la banda actual

# Crear figura

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

mne.viz.plot\_topomap(

valores, info, axes=ax, show=False, cmap="viridis",

sphere=(0.00, 0.0, 0.0, 0.09) # Ajusta el tamaño de la cabeza

)

\*Nota: viridis es el tipo de imagen (esta es la del mapa azul y amarillo) y sphere es la escala (normalmente 0.085 o 0.9

ax.set\_title(f"{banda}")

# Guardar la imagen

nombre\_imagen = os.path.join(carpeta\_salida, f"IMAGEN\_{i}\_{banda}.png")

plt.tight\_layout()

plt.savefig(nombre\_imagen, dpi=300)

plt.close()

print(f"Imagen guardada: {nombre\_imagen}")

Bandas código:

# Definir las bandas de frecuencia

bands = {

'Delta': [0.5, 4],

'Theta': [4, 8],

'Alpha': [8, 12],

'Beta': [12, 30],

'Gamma': [30, 50]

}

# Calcular la PSD directamente desde el objeto Raw usando compute\_psd

psds = raw\_clean.compute\_psd(fmin=0.5, fmax=50, n\_fft=2048)

freqs = psds.freqs

psd\_data = psds.get\_data()

# Calcular la potencia media en cada banda de frecuencia

band\_powers = {}

for band, (fmin, fmax) in bands.items():

idx\_band = np.logical\_and(freqs >= fmin, freqs <= fmax)

band\_power = np.mean(psd\_data[:, idx\_band], axis=1)

band\_powers[band] = band\_power

# Configurar los subplots: una fila por cada banda (5 bandas)

fig, axes = plt.subplots(1, len(bands), figsize=(15, 5))

# Generar el mapeo topográfico para cada banda de frecuencia en subplots

for ax, (band, power) in zip(axes, band\_powers.items()):

mne.viz.plot\_topomap(power, raw\_clean.info, cmap='viridis', axes=ax, show=False, vlim=(np.min(power), np.max(power)), contours=0)

ax.set\_title(f"{band} band")

# Mostrar la figura completa

plt.tight\_layout()

plt.show()

**Dibujo de colores

Descripción generada automáticamente con confianza media**

Generar un GIF O MP4:

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.animation import FuncAnimation

import mne

# Seleccionar los tiempos (ventanas) para la animación

times = np.arange(0, len(raw\_clean.times), 1000) # Ajusta el paso temporal según tus datos

# Crear la figura y el eje

fig, ax = plt.subplots()

# Función para actualizar la topografía en cada fotograma

def update\_topomap(frame):

ax.clear()

time = raw\_clean.times[frame]

data = raw\_clean.copy().crop(tmin=time, tmax=time + 1/raw\_clean.info['sfreq']).get\_data()

mne.viz.plot\_topomap(data[:, 0], raw\_clean.info, axes=ax, show=False, cmap='viridis')

ax.set\_title(f'Tiempo: {time:.2f} s')

# Crear la animación

anim = FuncAnimation(fig, update\_topomap, frames=times, interval=200, repeat=False)

# Guardar la animación como GIF o MP4

anim.save("activacion\_cerebral.gif", writer='imagemagick', fps=10) # Cambia a 'ffmpeg' si deseas MP4

plt.show()

**ANÁLISIS ESTADÍSTICO**

Test no paramétrico en eeg.

<https://www.sciencedirect.com/science/article/pii/S0165027007001707>

Maris, E., & Oostenveld, R. (2007). Nonparametric statistical testing of EEG-and MEG-data. Journal of neuroscience methods, 164(1), 177-190.