"""Run the default preprocessing pipeline on soarrKULee.

This script runs the necessary prereprocessing steps on the sparrKULee dataset, starting from the raw caches,

to arrive at the fully preprocessed files.

The raw caches are downloaded from the challenge website and should be placed in the folder specified by the

raw\_eeg\_dir variable.

The preprocessed EEG will be saved in the folder specified by the preprocessed\_eeg\_dir variable.

The caches have been synchronized with the stimulus data and should all have a length of 5 seconds.

On the raw caches, the following preprocessing steps have been performed:

(ie. synched with the stimulus data

and loaded into python)

You are free to use these caches for your own preprocessing pipeline,

however, if you want to use certain artifact steps, such as the artifact removal MWF,

be aware that the output from these steps might differ when using input windows of just 5 seconds.

If you want to use our artificat removal steps, we recommend to use the MWF caches,

on which the artifact removal MWF has already been performed, and not compute it yourself.

The MWF caches are saved at 1024Hz, see sparKULee\_mwf.py for more information on how to use these caches.

eeg\_steps = [

LinkStimulusToBrainResponse(

stimulus\_data=stimulus\_steps,

extract\_stimuli\_information\_fn=BIDSAPRStimulusInfoExtractor(),

grouper=BIDSStimulusGrouper(

bids\_root=root\_dir,

mapping={"stim\_file": "stimulus\_path", "trigger\_file": "trigger\_path"},

subfolders=["stimuli", "eeg"],

),

),

LoadEEGNumpy(unit\_multiplier=1e6, channels\_to\_select=list(range(64))),

AlignPeriodicBlockTriggers(biosemi\_trigger\_processing\_fn),

DefaultSave(raw\_eeg\_dir,

{'eeg': 'data'},

filename\_fn = bids\_filename\_fn,

clear\_output=True,

overwrite=overwrite),

]

"""

import argparse

import datetime

import gzip

import json

import logging

import os

import glob

from typing import Any, Dict, Sequence

import librosa

import numpy as np

import math

import scipy.signal.windows

from brain\_pipe.dataloaders.path import GlobLoader

from brain\_pipe.pipeline.default import DefaultPipeline

from brain\_pipe.preprocessing.brain.artifact import (

InterpolateArtifacts,

ArtifactRemovalMWF,

)

from brain\_pipe.preprocessing.brain.eeg.biosemi import (

biosemi\_trigger\_processing\_fn,

)

from brain\_pipe.preprocessing.brain.eeg.load import LoadEEGNumpy

from brain\_pipe.preprocessing.brain.epochs import SplitEpochs

from brain\_pipe.preprocessing.brain.link import (

LinkStimulusToBrainResponse,

BIDSStimulusInfoExtractor,

)

from brain\_pipe.preprocessing.brain.rereference import CommonAverageRereference

from brain\_pipe.preprocessing.brain.trigger import (

AlignPeriodicBlockTriggers,

)

from brain\_pipe.preprocessing.filter import SosFiltFilt

from brain\_pipe.preprocessing.resample import ResamplePoly

from brain\_pipe.preprocessing.stimulus.audio.envelope import GammatoneEnvelope

from brain\_pipe.preprocessing.stimulus.audio.spectrogram import LibrosaMelSpectrogram

from brain\_pipe.preprocessing.stimulus.load import LoadStimuli

from brain\_pipe.runner.default import DefaultRunner

from brain\_pipe.save.default import DefaultSave

# from mel import DefaultSave

from brain\_pipe.utils.log import default\_logging, DefaultFormatter

from brain\_pipe.utils.path import BIDSStimulusGrouper

from typing import Dict, Any, Sequence, Optional, Union, Mapping

import numpy as np

from brain\_pipe.pipeline.base import PipelineStep

class LoadEEGNumpyTest(PipelineStep):

"""Load EEG data.

This step uses MNE to load EEG data.

"""

def \_\_init\_\_(

self, keys={"data\_path": "data"}, copy\_data\_dict=False, \*mne\_args, \*\*mne\_kwargs

):

"""Create a new LoadEEG instance.

Parameters

----------

eeg\_path\_key: str

The key of the EEG path in the data dict.

eeg\_data\_key: str

The key of the EEG data in the data dict.

"""

super().\_\_init\_\_(copy\_data\_dict=copy\_data\_dict)

self.keys = self.parse\_dict\_keys(keys, "keys")

self.mne\_args = mne\_args

self.mne\_kwargs = mne\_kwargs

def \_\_call\_\_(self, data\_dict: Dict[str, Any]) -> Dict[str, Any]:

"""Load EEG data from a npy file.

Parameters

----------

data\_dict: Dict[str, Any]

The data dict containing the EEG path.

Returns

-------

Dict[str, Any]

The data dict with the EEG data and the EEG info.

"""

for from\_key, to\_key in self.keys.items():

path = data\_dict[from\_key]

# Support for gzipped files.

raw =np.load(path)

# swap axes

raw = np.swapaxes(raw, 0, 1)

data\_dict['data'] = raw

data\_dict['eeg\_key'] = os.path.basename(path)

data\_dict['data\_fs'] = 1024

return data\_dict

class BIDSAPRStimulusInfoExtractor(BIDSStimulusInfoExtractor):

"""Extract BIDS compliant stimulus information from an .apr file."""

def \_\_call\_\_(self, brain\_dict: Dict[str, Any]):

"""Extract BIDS compliant stimulus information from an events.tsv file.

Parameters

----------

brain\_dict: Dict[str, Any]

The data dict containing the brain data path.

Returns

-------

Sequence[Dict[str, Any]]

The extracted event information. Each dict contains the information

of one row in the events.tsv file

"""

event\_info = super().\_\_call\_\_(brain\_dict)

# Find the apr file

path = brain\_dict[self.brain\_path\_key]

apr\_path = "\_".join(path.split("\_")[:-1]) + "\_eeg.apr"

# Read apr file

apr\_data = self.get\_apr\_data(apr\_path)

# Add apr data to event info

for e\_i in event\_info:

e\_i.update(apr\_data)

return event\_info

def get\_apr\_data(self, apr\_path: str):

"""Get the SNR from an .apr file.

Parameters

----------

apr\_path: str

Path to the .apr file.

Returns

-------

Dict[str, Any]

The SNR.

"""

import xml.etree.ElementTree as ET

apr\_data = {}

tree = ET.parse(apr\_path)

root = tree.getroot()

# Get SNR

interactive\_elements = root.findall(".//interactive/entry")

for element in interactive\_elements:

description\_element = element.find("description")

if description\_element.text == "SNR":

apr\_data["snr"] = element.find("new\_value").text

if "snr" not in apr\_data:

logging.warning(f"Could not find SNR in {apr\_path}.")

apr\_data["snr"] = 100.0

return apr\_data

def test\_filename\_fn(data\_dict, feature\_name, set\_name=None):

"""Default function to generate a filename for the data.

Parameters

----------

data\_dict: Dict[str, Any]

The data dict containing the data to save.

feature\_name: str

The name of the feature.

set\_name: Optional[str]

The name of the set. If no set name is given, the set name is not

included in the filename.

Returns

-------

str

The filename.

"""

eeg\_key = data\_dict['eeg\_key']

return eeg\_key

def temp\_unpack\_data(data\_path):

data = dict(np.load(data\_path))

# save all keys, values in separate data path. TODO: fix this such that we don't have to do this

for key, value in data.items():

np.save(os.path.dirname(data\_path) + key + '.npy', value)

def run\_preprocessing\_pipeline(

root\_dir,

preprocessed\_eeg\_dir,

nb\_processes=4,

overwrite=False,

log\_path="sparrKULee.log",

):

"""Construct and run the preprocessing on SparrKULee.

Parameters

----------

root\_dir: str

The root directory of the dataset.

preprocessed\_eeg\_dir:

The directory where the preprocessed EEG should be saved.

nb\_processes: int

The number of processes to use. If -1, the number of processes is

automatically determined.

overwrite: bool

Whether to overwrite existing files.

log\_path: str

The path to the log file.

"""

#########

# PATHS #

#########

os.makedirs(preprocessed\_eeg\_dir, exist\_ok=True)

###########

# LOGGING #

###########

handler = logging.FileHandler(log\_path)

handler.setLevel(logging.DEBUG)

handler.setFormatter(DefaultFormatter())

default\_logging(handlers=[handler])

################

# DATA LOADING #

################

logging.info("Retrieving layout...")

data\_loader = GlobLoader(

[os.path.join(root\_dir, "sub\*.npy")],

filter\_fns=[],

key="data\_path",

)

#########################

# RUNNING THE PIPELINE #

#########################

logging.info("Starting with the EEG preprocessing")

logging.info("===================================")

eeg\_steps = [

LoadEEGNumpyTest(),

SosFiltFilt(

scipy.signal.butter(1, 0.5, "highpass", fs=1024, output="sos"),

emulate\_matlab=True,

axis=1,

),

InterpolateArtifacts(),

ArtifactRemovalMWF(),

CommonAverageRereference(),

ResamplePoly(64, axis=1),

DefaultSave(

preprocessed\_eeg\_dir,

{"eeg": "data"},

overwrite=overwrite,

clear\_output=True,

filename\_fn=test\_filename\_fn,

),

]

#########################

# RUNNING THE PIPELINE #

#########################

logging.info("Starting with the EEG preprocessing")

logging.info("===================================")

# Create data\_dicts for the EEG files

# Create the EEG pipeline

eeg\_pipeline = DefaultPipeline(steps=eeg\_steps)

DefaultRunner(

nb\_processes=nb\_processes,

logging\_config=lambda: None,

).run(

[(data\_loader, eeg\_pipeline)],

)

if \_\_name\_\_ == "\_\_main\_\_":

# Load the config

# get the top folder of the dataset

challenge\_folder = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

with open(os.path.join(challenge\_folder, 'util', 'config.json'), "r") as f:

config = json.load(f)

# Set the correct paths as default arguments

dataset\_folder = config["dataset\_folder"]

test\_folder = os.path.join(dataset\_folder, config["test\_folder"])

task = 'TASK1\_match\_mismatch' # [' TASK1\_match\_mismatch', 'TASK2\_regression']

preprocessed\_eeg\_folder = os.path.join(

test\_folder, task, f'{config["preprocessed\_eeg\_folder"]}'

)

raw\_eeg\_dir = os.path.join(test\_folder, task, 'raw\_eeg')

# unpack the data

raw\_eeg\_data = glob.glob(os.path.join(raw\_eeg\_dir, '\*\_eeg\_raw.npz'))

for data\_path in raw\_eeg\_data:

print(f'processing {data\_path}')

temp\_unpack\_data(data\_path)

default\_log\_folder = os.path.dirname(os.path.abspath(\_\_file\_\_))

# Parse arguments from the command line

parser = argparse.ArgumentParser(description="Preprocess the auditory EEG dataset")

parser.add\_argument(

"--nb\_processes",

type=int,

default=1,

help="Number of processes to use for the preprocessing. "

"The default is to use all available cores (-1).",

)

parser.add\_argument(

"--overwrite", action="store\_true", help="Overwrite existing files"

)

parser.add\_argument(

"--log\_path", type=str, default=os.path.join(

default\_log\_folder,

"sparrKULee\_{datetime}.log"

)

)

parser.add\_argument(

"--dataset\_folder",

type=str,

default=raw\_eeg\_dir,

help="Path to the folder where the dataset is downloaded",

)

parser.add\_argument(

"--preprocessed\_raw\_eeg\_path",

type=str,

default=preprocessed\_eeg\_folder,

help="Path to the folder where the preprocessed EEG will be saved",

)

args = parser.parse\_args()

# Run the preprocessing pipeline

run\_preprocessing\_pipeline(

args.dataset\_folder,

args.preprocessed\_raw\_eeg\_path,

args.nb\_processes,

args.overwrite,

args.log\_path.format(

datetime=datetime.datetime.now().strftime("%Y%m%d\_%H%M%S")

),

)