```
import os
import glob
import numpy as np
import pandas as pd
from tqdm import tqdm
from pathlib import Path
import matplotlib.pyplot as plt
from tensorflow.python.keras.utils.np_utils import to_categorical

from config import window_secs, painmonit_sensors, baseline_temp, sampling_rate_painmonit, num_repetitions_uzl, uzl_faulty, biosignals_dir
from scripts.data_handling import get_initials

#-----------------------------------------------------------------------------------------------------
# Functions PainMonit (UzL) dataset
#-----------------------------------------------------------------------------------------------------

def crossings_nonzero_neg2pos(data):
    """Function to find zero crossings (from negative to positive) in a signal.

    Parameters
    ----------
    data: numpy/list. The given numpy to search for zero crossings.

    Returns
    -------
    np: Crossings. Indices where crossings from negative to positive can be found.
    """
    npos =  data < 0
```

```
    return (npos[:-1] & ~npos[1:]).nonzero()[0]


def segment_uzl(df, baseline_shift= 5, plot= False):
    """Function to segment stim and baseline windows from synchronised 'PainMonit' data.

    Parameters
    ----------
    df: Panda. Synchronised PainMonit data of one subject.
    baseline_shift: Int. Number of seconds by which the "10 seconds baseline"-window is shifted backwards. Value describes the time between "baseline" and "stim". Default is 5.
    plot: Bool. Boolean to describe whether the segmentation process is plotted or not.
    """

    X = []
    y_heater = []
    y_covas = []

    # start of a stimuli
    stim = df["Heater_cleaned"] != baseline_temp
    stim[stim==False]=-1
    stim_starts = crossings_nonzero_neg2pos(stim.values)
    num_baseline_windows = 0

    # extract 10 seconds before stim
    window = int(window_secs * sampling_rate_painmonit)
    for start in stim_starts:
        # extract window before as baseline
        # move the baseline window about "baseline_shift" secs
        baseline_start = start - (baseline_shift * sampling_rate_painmonit)
```

```
                if (num_baseline_windows < num_repetitions_uzl) and (baseline_start > window) and (df["Heater_cleaned"].values[baseline_start - window: baseline_start]==baseline_temp).all():
                        X.append(df[painmonit_sensors].values[baseline_start - window: baseline_start])
                        y_covas.append(0)
                        y_heater.append(0)
                        num_baseline_windows += 1

                # extract window afterwards as stimulus
                start += 1
                temp = df["Heater_cleaned"].values[start]
                end = int(start + window)
                if (df["Heater_cleaned"].values[start: end]== temp).all():
                        X.append(df[painmonit_sensors].values[start: end])
                        y_covas.append(sum(df["COVAS"].values[start: end]))
                        y_heater.append(temp)

                if plot:
                        for name in ["Heater_cleaned", "Eda_RB"]:
                                data = df[name].values
                                data = (data-np.min(data))/(np.max(data)-np.min(data))
                                plt.plot(data)
                        plt.axvspan(baseline_start - window, baseline_start, facecolor='#2ca02c', alpha=0.5, label= "Baseline")
                        plt.axvspan(start, start + window, facecolor='#d62728', alpha=0.5, label= "Stim")
                        plt.legend()
                        plt.show()

        # --- Convert heater from temp to class
        # extract baseline + 1 one non painfull stim and 4 pain stim temperatures
```

```
    temps = np.unique(y_heater)

    conversion = { x : i for i, x in enumerate(temps)}
    # convert y from temperature to class label -> baseline:0, no pain stim: 1, pain stims: 2-5
    y_heater = np.vectorize(conversion.get)(y_heater)

    # --- Convert COVAS label into quartiles 0:No covas; 1: 1.Quartile, 2: 2. Quartile, [...], 4: 4.Quartile
    # Normalize values between 0 and 100
    y_covas = np.array(y_covas)
    y_covas = y_covas/y_covas.max()
    y_covas *= 100
    y_covas = np.array([int(x//(25)) + 1 if x > 0 else 0 for x in y_covas])
    y_covas[y_covas==5] = 4

    X = np.array(X)

    return X, y_heater, y_covas

def create_np_painmonit(original_dir = Path("datasets", "painmonit", "synchronised-data-files")):
    """Function to create np files of the painmonit dataset and save them.

    Parameters
    ----------
    original_data: Str. String to define the directory containing the synchronized PainMonit files (.csv).
    """

    if not Path(original_dir).exists():
```

```
        print(f"There is no directory '{original_dir.resolve()}'. Please place the datasets correctly.")
        return

    np_dir = Path(original_dir.parent, "np-dataset")
    if np_uzl_exists(np_dir):
        print(f"There is already a numpy dataset unter '{np_dir.resolve()}'. Dataset will not be overwritten.")
        return

    data_list = []
    heater_list = []
    covas_list = []
    subjects_list = []

    print("Create painmonit np dataset...")

    file_names = glob.glob(str(Path(original_dir, "*.csv")))
    # filter for faulty subjects
    file_names = [i for i in file_names if get_initials(i) not in uzl_faulty]

    # sort the list - lexicographically
    file_names.sort()

    for index, filename in enumerate(tqdm(file_names)):
        subject_data = pd.read_csv(filename, sep=";", decimal=",")

        X, y_heater, y_covas = segment_uzl(subject_data)
```

```
        data_list.append(X)
        heater_list.append(y_heater)
        covas_list.append(y_covas)
        subjects_list.append([index] * X.shape[0])

data = np.concatenate(data_list, axis=0)
heater = np.concatenate(heater_list, axis=0)
covas = np.concatenate(covas_list, axis=0)
subjects = np.concatenate(subjects_list, axis=0)

assert len(data)==len(heater)==len(covas)==len(subjects)

data = np.nan_to_num(data,)

# Data: Add channel axis
data = data[..., np.newaxis]

# Labels to categorical
heater = to_categorical(heater)
covas = to_categorical(covas)

if not np_dir.exists():
        os.makedirs(np_dir)

np.save(Path(np_dir, "X"), data)
np.save(Path(np_dir, "y_heater"), heater)
np.save(Path(np_dir, "y_covas"), covas)
np.save(Path(np_dir, "subjects"), subjects)
```

```
        print("\nData shape: ", data.shape)
        print("heater shape: ", heater.shape)
        print("covas shape: ", covas.shape)
        print("Subjects shape: ", subjects.shape)

        print(f"Np dataset created and saved under '{np_dir.resolve()}'.")

def np_uzl_exists(np_dir):
        """Function to check if np files of the UzL dataset already exist.

        Parameters
        ----------
        np_dir: string. String describing the location of the files.

        Returns
        -------
        bool: True if the dataset exists as np file, False otherwise.
        """

        data = Path(np_dir, "X.npy")
        heater = Path(np_dir, "y_heater.npy")
        covas = Path(np_dir, "y_covas.npy")
        subjects = Path(np_dir, "subjects.npy")

        return data.exists() and heater.exists() and covas.exists() and subjects.exists()

#-----------------------------------------------------------------------------------------------
# Functions biovid dataset
#-----------------------------------------------------------------------------------------------
```

```
def create_np_biovid(original_dir = Path("datasets", "biovid")):
    """Function to create np files of the BioVid dataset (PartA) and save them.

    Parameters
    ----------
    original_data: Str. String to define the directory containing the BioVid dataset.
    """

    np_dir = Path(original_dir, "np-dataset")
    if np_biovid_exists(np_dir):
        print(f"There is already a numpy dataset unter '{np_dir.resolve()}'. Dataset will not be overwritten.")
        return

    original_dir = Path(original_dir, "PartA")
    if not original_dir.exists():
        print(f"There is no directory '{original_dir.resolve()}'. Please place the datasets correctly.")
        return

    # load sample file
    sample_file = Path(original_dir, "starting_point", "samples.csv")
    if not sample_file.exists():
        print(f"BioVid - datset not found.\nThere must be a directory '{original_dir}' with the associated files, for example: '{sample_file}'")
        return

    data_list = []
    lable_list = []
```

```
        subjects_list = []


        print("Create BioVid np dataset...")
        samples = pd.read_csv(sample_file, delimiter='\t')


        for index, sample in tqdm(list(samples.iterrows())):
                # load current sample file
                curr_filename =  str(Path(original_dir, biosignals_dir, sample.subject_name,
sample.sample_name + '_bio.csv'))
                curr_data = pd.read_csv(curr_filename, delimiter='\t')


                data_list.append(curr_data)
                lable_list.append(sample.class_id)
                subjects_list.append(sample.subject_id)


        data = np.stack(data_list, axis= 0)
        lable = np.array(lable_list)
        subjects = np.array(subjects_list)


        assert len(data)==len(lable)==len(subjects)


        data = np.nan_to_num(data,)


        # Data: Add channel axis
        data = data[..., np.newaxis]


        # Labels to categorical
        lable = to_categorical(lable)
```

```
        if not Path(np_dir).exists():
                os.makedirs(np_dir)

        np.save(str(Path(np_dir, "X")), data)
        np.save(str(Path(np_dir, "y")), lable)
        np.save(str(Path(np_dir, "subjects")), subjects)

        print("\nData shape: ", data.shape)
        print("Lable shape: ", lable.shape)
        print("Subjects shape: ", subjects.shape)

        print(f"Np dataset created and saved under '{np_dir.resolve()}'.")

def np_biovid_exists(np_dir):
        """Function to check if np files of the BioVid dataset already exist.

        Parameters
        ----------
        np_dir: string. String describing the location of the files.

        Returns
        -------
        bool: True if the dataset exists as np file, False otherwise.
        """

        data = Path(np_dir, "X.npy")
        lable = Path(np_dir, "y.npy")

        return data.exists() and lable.exists()
```

```
#-----------------------------------------------------------------------------------------------
# Main
#-----------------------------------------------------------------------------------------------
if __name__ == "__main__":
        """Main function.
        """
        create_np_painmonit()
        create_np_biovid()
```