```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
# 0 - all logs shown
# 1 - filter out INFO logs
# 2 - additionally filter out WARNING logs
# 3 - additionally filter out ERROR logs

import platform
import numpy as np
import pandas as pd
import tensorflow as tf
from pathlib import Path
from scipy.stats import zscore
from scripts.augmentation import augment

from hcf import get_hcf, moving_average
from scripts.classifier import *
from scripts.preprocessing import remove_ecg_wandering, preprocess_np
from scripts.evaluation import loso_cross_validation, five_loso, accuracy, from_categorical, rfe_loso
from scripts.data_handling import read_biovid_np, pick_classes, normalize, resample_axis,
read_painmonit_np, normalize_features

from config import painmonit_sensors, biovid_sensors, sampling_rate_biovid, sampling_rate_painmonit

#-----------------------------------------------------------------------------------------------
# Functions
#-----------------------------------------------------------------------------------------------

def prepare_data(X, y, subjects, param):
```

```
        # remove ECG wandering in BioVid dataset
        if "ecg" in param["selected_sensors"]:
                ecg_index = param["selected_sensors"].index("ecg")
                X[:, :, ecg_index, :] = np.apply_along_axis(func1d= remove_ecg_wandering, axis= 1,
arr=X[:, :, ecg_index, :])


        if "preprocess" in param and param["preprocess"]:
                print("Preprocess signals...")
                X = preprocess_np(X, sensor_names= param["sensor_names"], sampling_rate=
param["resample"])
                print("Signals preprocessed.")


        # ---------------------------------------- HCF
        hcf = get_hcf(dataset= param["dataset"])


        # select sensors
        column_names = hcf.columns.values
        # All column names that start with sensor strings in "selected sensors"
        sensor_columns = [x for x in column_names for name in param["selected_sensors"] if
x.startswith(name)]
        # Select columns
        hcf = hcf[sensor_columns]


        if "hcf_norm" in param and param["hcf_norm"]:
                hcf = normalize_features(hcf)


        hcf = hcf.fillna(0)


        # ---------------------------------------- Raw
```

```
    if "aug" in param and type(param["aug"]) == list:
        aug = augment(X, y, l = param["aug"], dataset= param["dataset"])
    else:
        aug = None

    if "cut" in param and param["cut"] is not None:
        start = int(param["input_fs"] * param["cut"][0])
        end = int(param["input_fs"] * param["cut"][1])
        X = X[:, start:end]
        aug = aug[:, start:end] if aug is not None else None

    if "resample" in param and param["resample"] is not None:
        X = resample_axis(X, input_fs= param["input_fs"], output_fs= param["resample"])
        aug = resample_axis(aug, input_fs= param["input_fs"], output_fs= param["resample"]) if
aug is not None else None

    sensor_ids = [param["sensor_names"].index(x) for x in param["selected_sensors"]]
    X = X[:, :, sensor_ids, :]
    aug = aug[:, :, sensor_ids, :] if aug is not None else None

    if "smooth" in param and param["smooth"] != None:
        for s in range(X.shape[2]):
            X[:, :, s, :] = np.apply_along_axis(func1d= moving_average, axis= 1, arr=X[:, :, s,
:], w=param["smooth"])
            if aug is not None:
                aug[:, :, s, :] = np.apply_along_axis(func1d= moving_average, axis= 1,
arr=aug[:, :, s, :], w=param["smooth"])

    if "minmax_norm" in param and param["minmax_norm"]:
        X = normalize(X)
```

```
            aug = normalize(aug)
        if "znorm" in param and param["znorm"]:
            X = zscore(X, axis= 1)
            aug = zscore(aug, axis= 1)

        # ---------------------------------------- Generic
        # select classes
        if "classes" in param and param["classes"] is not None:
            # select certain classes from the data
            X, aug, hcf, subjects, y = pick_classes(data = [X, aug, hcf, subjects], y= y, classes =
param["classes"], input_is_categorical= True)

        return X, aug, hcf, y, subjects

def conduct_experiment(X, y, subjects, clf, name, five_times= False, rfe= False):
        """ Method to conduct an experiment. Data to perform a ML task needs to be given.

        Args:
            X_cur (dataframe): X.
            y_cur (dataframe): y.
            subjects_cur (dataframe): subjects vector.
            clf (classifer): clf to use.
            name (str): Name of the file to save.
            five_times (bool, optional): Whether to conduct a 5x mean experiment. Defaults to
False.
        """

        X, aug, hcf, y, subjects = prepare_data(X, y, subjects, clf.param)
```

```
	print("X shape after preprocessing: ", X.shape)
	print("HCF shape after preprocessing: ", hcf.shape)

	if rfe:
		return rfe_loso(X, aug, hcf, y, subjects, clf)
	else:
		if five_times:
			return five_loso(X, aug, hcf, y, subjects, clf, output_csv = Path("results", "5_loso_{}.csv".format(name)))
		else:
			return loso_cross_validation(X, aug, hcf, y, subjects, clf, output_csv = Path("results", "{}.csv".format(name)))


def check_simple_metrics():
	print("Checking simplistic metrics...")

	df = pd.DataFrame([])

	# read in biovid
	param = {
		"dataset": "biovid",
		"input_fs": sampling_rate_biovid,
		"sensor_names": biovid_sensors,
		"selected_sensors": ["gsr"],
		"classes": [[0], [4]],
	}
	x_biovid, y_biovid, subjects_biovid = read_biovid_np()
	x_biovid, _, hcf, y_biovid, subjects_biovid = prepare_data(x_biovid, y_biovid, subjects_biovid, param)
	x_biovid = x_biovid.copy() # make variable available inside functions that are defined here
```

```
    y_biovid = from_categorical(y_biovid)


    # read in painmonit
    param["dataset"]= "painmonit"
    param["input_fs"] = sampling_rate_painmonit
    param["sensor_names"] = painmonit_sensors
    param["painmonit_label"]= "heater"
    param["classes"]= [[0], [5]]
    param["selected_sensors"]= ["Eda_RB"]
    x_painmonit = None
    x_painmonit, y_painmonit, subjects_painmonit = read_painmonit_np(label=
param["painmonit_label"])
    x_painmonit, _, hcf, y_painmonit, subjects_painmonit = prepare_data(x_painmonit,
y_painmonit, subjects_painmonit, param)
    x_painmonit = x_painmonit.copy() # make variable available inside functions that are defined
here
    y_painmonit = from_categorical(y_painmonit)


    def evaluate_metric(metric_str):
        """Function to evaluate a metric on the painmonit and biovid dataset.
        The given `metric_str` will be applied in the template 'f"[{metric_str} for x in dataset[:, :,
0, 0]]"'.
        Results will be saved in the df at locations '["Painmonit", metric_str]' and '["Biovid",
metric_str]'.

        Args:
            metric_str (string): The metric to check. For example, "x[0] < x[-1]".
        """
        x_painmonit # make variable available for 'eval' call
        x_biovid # make variable available for 'eval' call
```

```
            # evaluate painmonit
            pred_painmonit = eval(f"[{metric_str} for x in x_painmonit[:, :, 0, 0]]")
            acc_painmonit = round(accuracy(pred_painmonit, y_painmonit) * 100, 2)

            # evaluate biovid
            pred_biovid = eval(f"[{metric_str} for x in x_biovid[:, :, 0, 0]]")
            acc_biovid = round(accuracy(pred_biovid, y_biovid) * 100, 2)

            # save results
            df.loc["Painmonit", metric_str] = acc_painmonit
            df.loc["Biovid", metric_str] = acc_biovid

    evaluate_metric(metric_str= "x[0] < x[-1]")
    evaluate_metric(metric_str= "len(x) * (7/10) < np.argmax(x)")
    evaluate_metric(metric_str= "len(x) * (1/3) < np.argmax(x) - np.argmin(x)")
    evaluate_metric(metric_str= "len(x) * (1/4) < np.argmax(x) - np.argmin(x)")
    evaluate_metric(metric_str= "0 < sum(x[1:] - x[:-1])")

    # save table
    print(df)
    df.to_csv(Path("results", "simple_metrics.csv"), sep= ";", decimal= ",")

def check_gpu():

    if 'linux' in platform.platform().lower():
        print("Check GPU...")
        if len(tf.config.list_physical_devices('GPU')) == 0:
            print("GPU is not available!")
```

```
                quit()

        print("GPU is available!")

if __name__ == "__main__":
    """Main function.
    """
    tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)

    # Simple metrics
    check_simple_metrics()

    #---------------------------------------------------------------------------------------------
    # Check if tensorflow is available
    #---------------------------------------------------------------------------------------------
    check_gpu()

    #---------------------------------------------------------------------------------------------
    # Configuration begin
    #---------------------------------------------------------------------------------------------
    param= {
        "dataset": "painmonit",
        "resample": 256, # Give sampling_rate to resample to
        "selected_sensors": ["Eda_RB"],
        "classes": [[0], [5]],
    }

    '''
    # biovid
```

```
param= {
        "dataset": "biovid",
        "resample": 256, # Give sampling_rate to resample to
        "selected_sensors": ["gsr"],
        "classes": [[0], [4]],
}
'''

sensor_names = []
#--------------------------------------------------------------------------------------------------
# Configuration end
#--------------------------------------------------------------------------------------------------

X, y, subjects = None, None, None

if param["dataset"] == "biovid":
        X, y, subjects = read_biovid_np()
        param["sensor_names"] = biovid_sensors
        param["input_fs"] = 512
elif param["dataset"] == "painmonit":
        param["painmonit_label"]= "heater" #or "covas"
        X, y, subjects = read_painmonit_np(label= param["painmonit_label"])
        param["sensor_names"] = painmonit_sensors
        param["input_fs"] = 250
else:
        print("""Dataset '{}' is not available.
        Please choose either 'biovid' or 'painmonit' and make sure the according np files are created correctly.
        """.format(param["dataset"]))
```

```
                quit()


        assert len(X)==len(y)==len(subjects)


        print("\nDataset shape:")
        print("X.shape")
        print(X.shape)
        print("y.shape")
        print(y.shape)
        print("subjects.shape")
        print(subjects.shape)
        print("\n")



        # HCF
        for selected_sensors  in [["Eda_RB"]]:
                for n_estimators in [50, 100, 150, 200, 250]:
                        param["n_estimators"] = n_estimators
                        param["selected_sensors"] = selected_sensors
                        conduct_experiment(X.copy(), y.copy(), subjects.copy(), clf= rf(param), name=
"rf", five_times= True)


        # Deep learning
        param.update({"epochs": 100, "bs": 32, "lr": 0.0001, "smooth": 256, "resample": 256,
"dense_out": 100, "minmax_norm": True})


        for clf in [mlp, cnn, cae, SCCAE, transformer, gaf_mdk]:
                try:
                        conduct_experiment(X.copy(), y.copy(), subjects.copy(), clf= clf(param.copy()),
name= param["dataset"], five_times= False, rfe= True)
```

```
except Exception as e:
    print(e)
```