Příloha 13

Kódové provedení detection_tools

original_method – vrátí nezměněné vstupy v podobě vhodné pro kalibraci
ratio_method – vrátí vstupy v poměrové podobě

```
import numpy as np
import pandas as pd
import warnings


def original_method(dataframe, nwa: int, nfa: int, prefix: str = ''):
    """
    Transform the rough data from copanda output into ratio realtions.
    peaks, wX_mean, wX_std, wX_Ra, wX_Skew, wX_Kurt, wX_rstd, wX_50,
wX_frq_n, wX_psd_n

    Aditionaly contains: category (if exists)

    Parameters
    ------------
    dataframe : pandas DataFrame
        Pandas dataframe with columns: peaks, wX_mean, wX_std, wX_Ra,
        wX_Skew, wX_Kurt, wX_rstd, wX_50, wX_frq_n, wX_psd_n
        for Actual and Bench values.

        Can be acquired from batched_tools copanda

    nwa : int
        Number of weights

    nfa : int
        Number of frequencies

    prefix : str
        Add prefix to default column names


    Returns
    ------------
    data_description : pandas DataFrame
        Contains ration of peaks, wX_mean, wX_std, wX_Ra,
        wX_Skew, wX_Kurt, wX_rstd, wX_50, wX_frq_n, wX_psd_n
        and aditionaly category


    """

    data_description = pd.DataFrame()

    # Learning entropy
    data_description['peaks'] = dataframe[f'{prefix}actual_peaks']
    data_description['peak_power'] =
dataframe[f'{prefix}actual_peak_power']

    # Mean and std
    for nw in range(nwa):
        actual_mean = dataframe[f'{prefix}actual_w{nw}_mean']
```

```
        actual_std = dataframe[f'{prefix}actual_w{nw}_std']
        data_description[f'w{nw}_mean'] = actual_mean
        data_description[f'w{nw}_std'] = actual_std

    # Secondary parameters
    for nw in range(nwa):
        actual_ra = dataframe[f'{prefix}actual_w{nw}_Ra']
        actual_skew = dataframe[f'{prefix}actual_w{nw}_Skew']
        actual_kurt = dataframe[f'{prefix}actual_w{nw}_Kurt']
        actual_rstd = dataframe[f'{prefix}actual_w{nw}_rstd']
        actual_50 = dataframe[f'{prefix}actual_w{nw}_50']

        data_description[f'w{nw}_rstd'] = actual_rstd
        data_description[f'w{nw}_50'] = actual_50
        data_description[f'w{nw}_Ra'] = actual_ra
        data_description[f'w{nw}_Skew'] = actual_skew
        data_description[f'w{nw}_Kurt'] = actual_kurt

    # Frequencies
    for nw in range(nwa):
        for nf in range(nfa):
            data_description[f'w{nw}_frq_{nf}'] =
dataframe[f'{prefix}w{nw}_frq_{nf}']
            data_description[f'w{nw}_psd_{nf}'] =
dataframe[f'{prefix}w{nw}_psd_{nf}']

    # Catagory - Error, Not Error
    if ('error_strat' in dataframe.columns):
        data_description['category'] = dataframe['error_strat'].cat.codes
        data_description['category_name'] = dataframe['error_strat']
    elif ('category' in dataframe.columns):
        data_description['category'] = dataframe['category'].cat.codes
        data_description['category_name'] = dataframe['category']

    return data_description


def ratio_method(dataframe, nwa: int, nfa: int, prefix: str = ''):
    """
    Transform the rough data from copanda output into ratio realtions.
    peaks, wX_mean, wX_std, wX_Ra, wX_Skew, wX_Kurt, wX_rstd, wX_50,
wX_frq_n, wX_psd_n

    Aditionaly contains: category (if exists)

    Parameters
    ------------
    dataframe : pandas DataFrame
        Pandas dataframe with columns: peaks, wX_mean, wX_std, wX_Ra,
        wX_Skew, wX_Kurt, wX_rstd, wX_50, wX_frq_n, wX_psd_n
        for Actual and Bench values.

        Can be acquired from batched_tools copanda

    nwa : int
        Number of weights

    nfa : int
        Number of frequencies

    prefix : str
```

```
        Add prefix to default column names


    Returns
    ------------
    data_description : pandas DataFrame
        Contains ration of peaks, wX_mean, wX_std, wX_Ra,
        wX_Skew, wX_Kurt, wX_rstd, wX_50, wX_frq_n, wX_psd_n
        and aditionaly category


    """

    data_description = pd.DataFrame(index=[0])

    # Learning entropy
    dataframe.loc[dataframe[f'{prefix}bench_peaks'] == 0,
f'{prefix}bench_peaks'] = 0.1
    dataframe.loc[dataframe[f'{prefix}actual_peaks'] == 0,
f'{prefix}actual_peaks'] = 0.1

    data_description['peaks_diff'] =
np.round(((dataframe[f'{prefix}actual_peaks']-
dataframe[f'{prefix}bench_peaks'])/dataframe[[f'{prefix}bench_peaks',
f'{prefix}actual_peaks']].min(axis=1))[0], 3)
    #data_description['peak_power_diff'] =
np.round((dataframe['mean_actual_peak_power']-
dataframe['mean_bench_peak_power'])/dataframe[['mean_bench_peak_power','mea
n_actual_peak_power']].min(axis=1), 3)


    # Mean and std
    for nw in range(nwa):
        bench_mean = dataframe[f'{prefix}bench_w{nw}_mean']
        actual_mean = dataframe[f'{prefix}actual_w{nw}_mean']
        bench_std = dataframe[f'{prefix}bench_w{nw}_std']
        actual_std = dataframe[f'{prefix}actual_w{nw}_std']
        data_description[f'w{nw}_mean_ratio'] = actual_mean/bench_mean
        data_description[f'w{nw}_std_ratio'] = actual_std/bench_std

    # Secondary parameters
    for nw in range(nwa):
        bench_ra = dataframe[f'{prefix}bench_w{nw}_Ra']
        actual_ra = dataframe[f'{prefix}actual_w{nw}_Ra']
        bench_skew = dataframe[f'{prefix}bench_w{nw}_Skew']
        actual_skew = dataframe[f'{prefix}actual_w{nw}_Skew']
        bench_kurt = dataframe[f'{prefix}bench_w{nw}_Kurt']
        actual_kurt = dataframe[f'{prefix}actual_w{nw}_Kurt']
        bench_rstd = dataframe[f'{prefix}bench_w{nw}_rstd']
        actual_rstd = dataframe[f'{prefix}actual_w{nw}_rstd']
        bench_50 = dataframe[f'{prefix}bench_w{nw}_50']
        actual_50 = dataframe[f'{prefix}actual_w{nw}_50']

        data_description[f'w{nw}_rstd'] = (actual_rstd -
bench_rstd)/dataframe[[f'{prefix}bench_w{nw}_rstd',
f'{prefix}actual_w{nw}_rstd']].min(axis=1)
        data_description[f'w{nw}_50_ratio'] = (actual_50 -
bench_50)/dataframe[[f'{prefix}bench_w{nw}_50',
f'{prefix}actual_w{nw}_50']].min(axis=1)
        data_description[f'w{nw}_Ra_ratio'] = (actual_ra -
bench_ra)/dataframe[[f'{prefix}bench_w{nw}_Ra',
```

```
f'{prefix}actual_w{nw}_Ra']].min(axis=1)
        data_description[f'w{nw}_Skew_ratio'] = (actual_skew -
bench_skew)/dataframe[[f'{prefix}bench_w{nw}_Skew',
f'{prefix}actual_w{nw}_Skew']].min(axis=1)
        data_description[f'w{nw}_Kurt_ratio'] = (actual_kurt -
bench_kurt)/dataframe[[f'{prefix}bench_w{nw}_Kurt',
f'{prefix}actual_w{nw}_Kurt']].min(axis=1)

    # Frequencies
    for nw in range(nwa):
        for nf in range(nfa):
            data_description[f'w{nw}_frq_{nf}'] =
dataframe[f'{prefix}w{nw}_frq_{nf}']
            data_description[f'w{nw}_psd_{nf}'] =
dataframe[f'{prefix}w{nw}_psd_{nf}']

    # Catagory - Error, Not Error
    if ('error_strat' in dataframe.columns):
        data_description['category'] = dataframe['error_strat'].cat.codes
        data_description['category_name'] = dataframe['error_strat']
    elif ('category' in dataframe.columns):
        data_description['category'] = dataframe['category'].cat.codes
        data_description['category_name'] = dataframe['category']

#data_description.loc[data_description['category_name']=='no_noise',
'category'] = 0

#data_description.loc[data_description['category_name']!='no_noise',
'category'] = 1
    return data_description
```