MEMD_Analysis_iEEG

Table of contents

General info

This repository contains python code for generating the main figures of the paper ""

Setup

Choose the name of the folder *results*

After downloading the project from Github, you can start producing results and figures. Results will be stored in a folder *results* within the main directory folder of the project by default. However, the user can change the name of the folder by going to the python file `results.py` which is located in the following path: `MEMD_Analysis_iEEG\funcs\Global_settings\results.py` The output of each python script by default will be saved in a folder named exactly as the python script file running within the ***results*** folder.

Main Figures

In this section we provide the code for computing and generating the some of the main figures appear in the paper.

Seizure Dissimilarity and Seizure IMF Distance Heatmaps

In order to generate the heatmap plots of seizure dissimilarity and seizure IMF distances, you need to run the following python scripts:

- funcs_need/`sz_dist_raw.py`

- main_figures/Heatmaps_sz_diss_dist_raw.py

Plots generated only for patients with more than 5 seizures

Seizure Dissimilarity and Seizure IMF Distance Heatmaps (standardised)

In order to generate the heatmap plots of standardised seizure dissimilarity and standardised seizure IMF distances, you need to run the following python scripts:

- funcs_need/sz_dist_raw.py
- funcs_need/sz_dist_stand_raw.py
- main_figures/Heatmaps_sz_diss_dist_stand_raw.py

Plots generated only for patients with more than 5 seizures

Marginal Hilbert Spectrum of IMFs

For obtaining the graphical representations, the following python scripts need to be run:

- funcs_need/Hilbert_output.py
- main_figures/PSD_computation.py

Plots generated for all patients

Contribution of iEEG main frequency bands to the 24h IMF

For obtaining the graphical representations, the following python scripts need to be run:

- funcs_need/Hilbert_output.py
- main_figures/PSD_computation.py
- funcs_need/Relative_power_24IMF.py
- main_figures/Relative_power_24IMF_allP.py

Plots generated for all patients

Gini index of IMFs for every frequency band across all subjects

For obtaining the graphical representation, the following python scripts need to be run:

- funcs_need/Hilbert_output.py
- main_figures/PSD_computation.py
- funcs_need/Gini_index_allIMFs.py
- main_figures/Gini_index_allIMFs_allP.py

Plots generated for all patients

A combination of IMF seizure distances on different timescales can explain seizure dissimilarity in most subjects in a multiple regression model

- funcs_need/sz_dist_raw.py
- funcs_need/sz_dist_stand_raw.py
- funcs_need/Preparing_data_for_LASSO.py
- funcs_need/Constrained_LASSO_IMFs.py
- funcs_need/LinearRegression_IMFs.py
- main_figures/Gather_Rsquared_allP.py

Plots generated only for patients with more than 5 seizures

Supporting figures

Scatterplots of seizure dissimilarity and seizure distances

In order to obtain the scatterplots between seizure dissimilarity and seizure IMF distance, as well as the ones of seizure dissimilarity and seizure time distance, you will have to run the following python scripts:

- funcs_need/sz_dist_raw.py
- funcs_need/Mantel_test_raw.py

The output of Mantel_test_raw.py would be the aforementioned scatterplots (one for the time distance and multiple ones for all IMFs), along with the spearman correlation values displayed in the title of the plots. Mantel test results are also generated, but not displayed in the title of the scatterplots.

Plots generated only for patients with more than 5 seizures

Run for one patient or all at once

In most of the python files there is the choice of generating the results for 1 or more patients using parallel programming. This option is available through the following code chunk that appears at the end of the python file/files:

```
def parallel_process ():
    processed = 0

    folders = os.listdir ( os.path.join ( ROOT_DIR, input_path ) )
    files = [os.path.join ( ROOT_DIR, input_path, folder ) for folder in folders]

    start_time = time.time ()
    # Test to make sure concurrent map is working
    with ProcessPoolExecutor ( max_workers=4 ) as executor:
        futures = [executor.submit ( process_file, in_path ) for in_path in files]
        for future in as_completed ( futures ):
            if future.result() == True:
               processed += 1
               print ( "Processed {}files.".format ( processed, len ( files ) ),
end="\r" )

    end_time = time.time ()
    print ( "Processed {} files in {:.2f} seconds.".format ( processed, end_time -
```

```
start_time ) )
```

For generating results for **one patient**, such as **patient ID06**, the user should add the following line of code:

```
files = files[5:6]
```

The above code chunk will look like this:

```
def parallel_process ():
    processed = 0

    folders = os.listdir ( os.path.join ( ROOT_DIR, input_path ) )
    files = [os.path.join ( ROOT_DIR, input_path, folder ) for folder in folders]

    files = files[5:6]

    start_time = time.time ()
    # Test to make sure concurrent map is working
    with ProcessPoolExecutor ( max_workers=4 ) as executor:
        futures = [executor.submit ( process_file, in_path ) for in_path in files]
        for future in as_completed ( futures ):
            if future.result() == True:
               processed += 1
               print ( "Processed {}files.".format ( processed, len ( files ) ),
end="\r" )

    end_time = time.time ()
    print ( "Processed {} files in {:.2f} seconds.".format ( processed, end_time -
start_time ) )
```

For running the python script/s for **more than one subject**, for example **subjects ID02, ID03, ID04, ID05**, the user needs to add the following line of code:

```
files = [files[i] for i in [1,2,3,4]]
```

The code chunk will look like this:

```
def parallel_process ():
    processed = 0

    folders = os.listdir ( os.path.join ( ROOT_DIR, input_path ) )
    files = [os.path.join ( ROOT_DIR, input_path, folder ) for folder in folders]
```

```
    files = [files[i] for i in [1,2,3,4]]

    start_time = time.time ()
    # Test to make sure concurrent map is working
    with ProcessPoolExecutor ( max_workers=4 ) as executor:
        futures = [executor.submit ( process_file, in_path ) for in_path in files]
        for future in as_completed ( futures ):
            if future.result() == True:
               processed += 1
               print ( "Processed {}files.".format ( processed, len ( files ) ),
end="\r" )

    end_time = time.time ()
    print ( "Processed {} files in {:.2f} seconds.".format ( processed, end_time -
start_time ) )
```

The above code chunk is not included in the following python scripts:

- FDR_Mantel_test_raw.py