Machine Learning-Based Seismic Signal Discriminator and Station Association Method

Kodai Sagae[1], Suguru Yabe[1], and Takahiko Uchide[1]

[1]Research Institute of Earthquake and Volcano Geology, Geological Survey of Japan, AIST

1. Abstract

This document provides weight parameters of a machine-learning model **DiET**, which is designed to classify signals as earthquake, tectonic tremor (hereafter referred to as tremor), or noise. It also includes the corresponding Python code. In addition, it provides Python code for **GrASP**, a method for associating detection results from multiple stations. For details on the methodology, please refer to Sagae et al. (2025, preprint).

2. Environment setup

This program is written in Python and requires the following packages.

- `python (3.11+)`
- `tensorflow (2.15.0)`
- `keras (2.15.0)`
- `numpy`
- `scipy`
- `scikit-learn`
- `pandas`
- `pyproj`
- `matplotlib`

Please unzip the attached zip file (`GSJ_open.zip`) and create the `GSJ_open` directory. We recommend using `Conda` to set up your environment. Users can build a virtual environment for this program by executing the following commands in the `Shell`. These commands use the `environment.yml` file located in the `GSJ_open` directory.

```
# Please move to the `GSJ_open` directory
conda env create -n [environment name] -f environment.yml
conda activate [environment name]
```

3. How to use

Python test codes for **DiET** and **GrASP** are included in the `GSJ_open/Examples` directory. The `DiET` and `GrASP` packages in the `Methods` directory contain the functions used in these Python test codes. The following sections describe usage examples for each Python test code.

3.1. Discriminator for Earthquake and Tremor (DiET)

DiET is a machine learning model that takes three-component spectrograms as inputs and outputs the signal probabilities for earthquakes, tremors, and noise, respectively. First, we provide Python code for spectrogram creation below.

`Show_spectrogram.py`

```
# Please move to the `Examples/DiET` directory
import sys
sys.path.append("../../Methods")
from DiET import Make_Spec

# Make spectrogram
Spec_scale, t_sample, f_sample = Make_Spec(Waveform_data)
```

Users can calculate a spectrogram (`Spec_scale`) by inputting one-minute waveform data (`Waveform_data`), with the instrument response removed, into the `Make_Spec()` function. By executing the `Show_spectrogram.py` script located in the `Examples/DiET` directory, users can calculate and display a spectrogram of randomly generated data. We describe parameters for spectrogram creation within `Params_DiET.py` located in the `Methods/DiET` directory. Except for the

sampling frequency of waveform data (`sampling_freq`), all parameters must remain unchanged.

In the next step, we show Python code to load and apply the **DiET** model.

`Model_apply.py`

```
import numpy as np
import sys
sys.path.append("../../Methods")
from DiET import Params_DiET, get_model

# Load model
model = get_model()

Station_number = 2
# Spectrogram with all pixel values of 1 is predicted as noise.
Input_data=np.ones((Station_number, Params_DiET.frequency_size,
Params_DiET.time_size, Params_DiET.num_channels))
# Remain only the spectrogram power in the 2-8 Hz.
Input_data[0,25:73,:,:] = 0.2

# Model prediction
pred_result = model.predict(Input_data, batch_size =
Station_number, verbose = 0)
```

Users can load the **DiET** model using the `get_model()` function. When performing prediction using this model, users input the followings.

- `Input_data`: A 4th-order tensor with following dimensions: the number of stations, frequency (73), time (71), and velocity components (3). The velocity components should be ordered as VX, VY, and VZ, with the vertical component recommended to be assigned to VZ.
- `batch_size`: The number of stations.

In the predictions (`pred_result`), the 1st, 2nd, and 3rd columns represent the signal probabilities for **earthquake**, **tremor**, and **noise**, respectively. By executing the `Model_apply.py` script located in the `Examples/DiET` directory, you can output the signal probabilities for the pseudo data.

3.2. Graph-based Associator with Signal Probability (GrASP)

GrASP is a method for extracting station groups where signals are detected, based on the signal probabilities at each station. To prepare for using **GrASP**, users need to create a station location information file in the following format:

- 1st column: Station name
- 2nd column: Latitude [deg]
- 3rd column: Longitude [deg]

Specifically, please refer to `Station_info.csv` in the `Data` directory. Additionally, users need to save the signal probability outputs from the **DiET** model in the following format:

- 1st column: Year
- 2nd column: Month
- 3rd column: Day
- 4th column: Hour
- 5th column: Minute
- From the 6th column onward: Signal probability for each station, in the same row order as in `Station_info.csv`.

Specifically, please refer to `Probability_T.csv` and `Probability_EQ.csv` in the `Data/Probability` directory.

We show Python code for associating tremor-detected stations using **GrASP**.

`GrASP_Tremor.py`

```
# Please move to the `Examples/GrASP` directory
import numpy as np
import pandas as pd
```

```
 import pyproj
 import sys
 sys.path.append("../../Methods")
 from GrASP import Params_GrASP, kNN_graph, Make_Stadis_array,
Apply_Tremor

 # Load Station location
 Station_loc = pd.read_csv('../../Data/Station_info.csv')
 Station_loc_data = Station_loc.values

 # Make adjacency matrix (permanent adjacency)
 Stadis_array_original = Make_Stadis_array(Station_loc_data)
 Stadis_array = kNN_graph(Stadis_array_original, k_neareast =
 Params_GrASP.k_neareast)

 # Converted to XY coordinates
 transformer=pyproj.Transformer.from_crs(Params_GrASP.EPSG_base,
 Params_GrASP.EPSG_proj, always_xy=True)
 Station_locx,Station_locy=transformer.transform(Station_loc_dat
 a[:, 2], Station_loc_data[:, 1])
 Station_loc_data_XY = np.vstack((Station_loc_data[:, 0],
 Station_locx, Station_locy))
 Station_loc_data_XY = Station_loc_data_XY.T

 # Load Tremor probability
 Read_Detection=pd.read_csv('../../Data/Probability/Probability_
T.csv')
 Data = Read_Detection.values

 # GrASP for Tremor
 Save_Candidate = Apply_Tremor(Data, Stadis_array,
Station_loc_data_XY)
```

The `Apply_Tremor()` function can be used to perform **GrASP** for tremor detection. The required input data are as follows:

- `Data`: Load the tremor probabilities from `Probability_T.csv`.
- `Stadis_array`: Adjacency matrix obtained by inputting a distance matrix to the `kNN_graph()` function. The distance matrix is calculated by inputting `Station_info.csv` to the `Make_Stadis_array()` function.
- `Station_loc_data_XY`: Station locations in a cartesian coordinate system.

Since the distance matrix (`Stadis_array_original`) is used multiple times, we recommend saving it as a separate file (`Data/Stadis_array_original.csv`). The output, `Save_Candidate`, follows the same format as `Probability_T.csv` and lists extracted station groups. Users can execute this process using the `GrASP_Tremor.py` script located in the `Examples/GrASP` directory.

Next, we show Python code for associating earthquake-detected stations using **GrASP**.

`GrASP_EQ.py`

```
import sys
sys.path.append("../../Methods")
from GrASP import Params_GrASP, kNN_graph, Apply_Nocut

# Loading EQ probability
Read_Detection=pd.read_csv('../../Data/Probability/Probability_EQ.csv')
Data = Read_Detection.values

# GrASP for EQ
Save_Candidate = Apply_Nocut(Data, Stadis_array)
```

The `Apply_Nocut()` function can be used to perform **GrASP** without graph partitioning. The required input data are as follows:

- `Data`: Load the earthquake probabilities from `Probability_EQ.csv`.
- `Stadis_array`: Same as the matrix used in the `Apply_Tremor()` function.

The output, `Save_Candidate`, follows the same format as `Probability_EQ.csv` and lists extracted station groups. The **`Apply_Nocut()`** function is also designed to accept tremor probabilities as input. When applying **GrASP** to a different seismic network, we recommend providing tremor probabilities over a certain period as validation data to the **`Apply_Nocut()`** function. Subsequently, you should examine the characteristics of the resulting station groups (e.g., Graph std: spatial spread of the stations, please refer to Sagae et al. (2025, preprint)). Based on these characteristics, parameters (described later) can be adjusted accordingly. Once this setup is complete, you can perform continuous analysis using the **`Apply_Tremor()`** function. Users can execute this process using the `GrASP_EQ.py` script located in the `Examples/GrASP` directory.

Finally, we show how to classify tremor-detected station groups into two categories.

`GrASP_Category.py`

```
import sys
sys.path.append("../../Methods")
from GrASP import Params_GrASP, kNN_graph, Apply_Categorization

# Load Tremor and EQ association results
Tremor_Association=pd.read_csv('../../Results/Tremor_candidate.
csv')
Data_T = Tremor_Association.values
EQ_Association = pd.read_csv('../../Results/EQ_candidate.csv')
Data_EQ = EQ_Association.values

# GrASP for Tremor Categorization
Save_result = Apply_Categorization(Stadis_array, Data_T,
Data_EQ)
```

Users can assign categories to tremor-detected station groups using the **`Apply_Categorization()`** function. The required input data are as follows:

- `Stadis_array`: Same as used in the `Apply_Tremor()` function.
- `Data_T` and `Data_EQ`: Association results for tremor and earthquake.

The output, `Save_result`, has a category added to the last column of `Data_T`, with Category 1 indicating probable tremor and Category 2 indicating possible tremor. Users can execute this process using the `GrASP_Category.py` script located in the `Examples/GrASP` directory.

We describe the parameters for **GrASP** within `Params_GrASP.py` located in the `Methods/GrASP` directory. When changing the application area of **GrASP** from the Japan Trench, users will need to adjust the following parameters:

- `EPSG_proj`: Change to the EPSG code of the UTM coordinate system that includes the target area (refer to resources such as https://spatialreference.org/ for the EPSG code).
- `k_nearest`: Adjust according to the number of stations. It is recommended to set this value to approximately $\sqrt{N}$, where N is the number of stations.
- `Std_thres`: This is a teleseismic event removal parameter. Adjust it based on the characteristics of the station groups extracted from the validation data using the `Apply_Nocut()` function. Following Sagae et al. (2025, preprint), we recommend setting this value to 3σ of Graph_std, which represent spatial spread of stations.
- `Max_eigen`: Threshold related to the cumulative contribution rate of the second smallest eigenvalue. Adjustment may be necessary if graph partitioning occurs frequently.

4. Disclaimer

- We have tested this program on Ubuntu 22.04. We cannot guarantee that it will work properly in other environments.
- The Geological Survey of Japan, AIST is not responsible or liable for any damages that may result from the use of this program.

5. Development Team

Key members

- Principal developer: Kodai Sagae (k.sagae@aist.go.jp)
- Code review, Draft check: Suguru Yabe (s.yabe@aist.go.jp)
- Project administrator: Takahiko Uchide (t.uchide@aist.go.jp)

Contributor

- Quick review: Masayuki Kano

6. Terms of use

- Please cite the following if you use this program:

Kodai Sagae, Suguru Yabe, and Takahiko Uchide (2025) Machine Learning-Based Seismic Signal Discriminator and Station Association Method, Open-File Report of Geological Survey of Japan, AIST, no. 767.

- Please contact the development team to report bugs.

7. Acknowledgements

In developing the **DiET** model, we used the velocity waveform data from the Seafloor Observation Network for Earthquakes and Tsunamis along the Japan Trench (S-net), operated by the National Research Institute for Earth Science and Disaster Resilience (NIED). We used hypocenter and phase data in the JMA Unified Earthquake Catalog. This research was supported by JSPS KAKENHI Grant Number JP21H05205, JP21H05203 in Grant-in-Aid for Transformative Research Areas (A) “Science of Slow-to-Fast Earthquakes”.

8. Licence

9. Reference

Kodai Sagae, Masayuki Kano, Suguru Yabe, Takahiko Uchide, 2025 (preprint), Machine Learning-Based Detection and Localization of Tectonic Tremors in the Japan Trench. *ESS Open Archive* . February 19, 2025.

DOI: 10.22541/essoar.174000875.59692909/v1