saipy.data.base.STEAD(directory = os.getcwd(), metadata_only = False)

A STEAD dataset object.

Parameters: directory (str, *optional*): path to directory where 'STEAD' folder can be found; defaults to current working directory

metadata_only (*bool, optional*): if True only metadata is read to the object

Attributes: metadata: pandas.DataFrame containing metadata

waveforms: h5py object containing waveforms

saipy.data.base.STEAD. trace_list()

A list of traces included in the STEAD dataset object.

Returns: list of traces

saipy.data.base. STEAD. distribution(parameter, traces = None, log = False, ax = None, color = 'slategrey')

Function for plotting a distribution of parameters according to metadata.

Parameters: parameter (*str*): the parameter whose distribution is to be plotted, make sure this has the same name as the corresponding columns of the metadata

traces (*list, optional*): if provided this plots the distribution only for the selected list of traces

log (*bool, optional*): if True plots the distribution in log scale

ax (*matplotlib.axes object, optional*): the axes on which the histogram is to be plotted

color (*str, optional*): defines the color of the bars in the resulting histogram; must be a valid matplotlib color.

saipy.data.base.STEAD.get_creime_data(traces = None)

Converts waveforms in the dataset to a format suitable for CREIME model (https://doi.org/10.1029/2022JB024595)

Parameters: traces (*list, optional*): if provided, returns data only for the selected list of traces

Returns: Xarr: numpy array of 3 component waveforms of length 512 samples.

yarr: numpy array of corresponding y-labels in format specified in https://doi.org/10.1029/2022JB024595

saipy.data.base.STEAD.get_polarcap_data(traces = None)

Converts waveforms in the dataset to a format suitable for PolarCAP model (https://doi.org/10.1016/j.aiig.2022.08.001)

Parameters: traces (*list, optional*): if provided, returns data only for the selected list of traces

Returns: Xarr: numpy array consisting 64 sample windows of the vertical component centered around the P-arrival time

saipy.data.base.STEAD.get_creime_rt_data(traces = None , training = False)

Converts waveforms in the dataset to a format suitable for CREIME_RT model

Parameters: traces (*list, optional*): if provided, returns data only for the selected list of traces

training (*bool, optional*): if True, returns data in the format suitable for training the model (this includes spectrograms)

Returns: Xarr: list of 3 component waveforms of random length

yarr: numpy array of corresponding y-labels in format used for CREIME_RT model

saipy.data.base.STEAD.get_dynapicker_data()

Returns data in format suitable for DynaPicker_v2, a improved model of DynaPicker (https://doi.org/10.48550/arXiv.2211.09539)

Returns: metadata: pandas.DataFrame containing metadata

waveforms: h5py object containing waveforms

saipy.data.base.INSTANCE(directory = os.getcwd(), metadata_only = False, data = 'both')

An INSTANCE dataset object.

Parameters: directory (str, *optional*): path to directory where 'INSTANCE' folder can be found; defaults to current working directory

metadata_only (*bool, optional*): if True only metadata is read to the object

data (*str, optional*): can be one of the following- 'both' (both noise and events data is included), 'noise' (only noise data is included) and 'events' (only events data is included)

Attributes: metadata_n: pandas.DataFrame containing metadata for noise

metadata_ev: pandas.DataFrame containing metadata for events

waveforms_n: h5py object containing noise waveforms

waveforms_ev: h5py object containing event waveforms

saipy.data.base.INSTANCE.trace_list_noise()

A list of noise traces included in the INSTANCE dataset object.

Returns: list of noise traces

saipy.data.base.INSTANCE.trace_list_events()

A list of event traces included in the INSTANCE dataset object.

Returns: list of event traces

saipy.data.base.INSTANCE.distribution(parameter, traces = None, log = False, ax = None, color = 'slategrey')

Function for plotting a distribution of parameters according to events metadata.

Parameters: parameter (*str*): the parameter whose distribution is to be plotted, make sure this has the same name as the corresponding columns of the metadata

traces (*list, optional*): if provided this plots the distribution only for the selected list of traces

log (*bool, optional*): if True plots the distribution in log scale

ax (*matplotlib.axes object, optional*): the axes on which the histogram is to be plotted

color (*str, optional*): defines the color of the bars in the resulting histogram; must be a valid matplotlib color.

saipy.data.base.INSTANCE.get_creime_data(traces_n = None, traces_ev = None)

Converts waveforms in the dataset to a format suitable for CREIME model (https://doi.org/10.1029/2022JB024595)

Parameters: traces_n (*list, optional*): if provided, returns data only for the selected list of noise traces

traces_ev (*list, optional*): if provided, returns data only for the selected list of event traces

Returns: Xarr: numpy array of 3 component waveforms of length 512 samples.

yarr: numpy array of corresponding y-labels in format specified in https://doi.org/10.1029/2022JB024595

saipy.data.base.INSTANCE.get_polarcap_data(traces = None, training = False)

Converts waveforms in the dataset to a format suitable for PolarCAP model (https://doi.org/10.1016/j.aiig.2022.08.001)

Parameters: traces (*list, optional*): if provided, returns data only for the selected list of traces

training (*bool, optional*): if True, returns data in the format suitable for training the model

Returns: Xarr: numpy array consisting 64 sample windows of the vertical component centered around the P-arrival time

yarr: numpy array containing ‘trace_polarity’ according to metadata

saipy.data.base.INSTANCE.get_creime_rt_data(traces_n = None, traces_ev = None, training = False)

Converts waveforms in the dataset to a format suitable for CREIME_RT model

Parameters: traces_n (*list, optional*): if provided, returns data only for the selected list of noise traces

traces_ev (*list, optional*): if provided, returns data only for the selected list of event traces

training (*bool, optional*): if True, returns data in the format suitable for training the model (this includes spectrograms)

Returns: Xarr: list of 3 component waveforms of random length

yarr: numpy array of corresponding y-labels in format used for CREIME_RT model

saipy.data.base.INSTANCE.get_dynapicker_data(event_type='EQ')

Returns data in format suitable for DynaPicker_v2, an improved version of DynaPicker model (https://doi.org/10.48550/arXiv.2211.09539)

Parameters: event_type (*str, optional*): if 'EQ' (default), returns data corresponding to events, else returns data corresponding to noise

Returns: metadata: pandas.DataFrame containing metadata

waveforms: h5py object containing waveforms

Real Data

saipy.data.realdata.waveform_download(wsp, net, sta, loc, chan, starttime, endtime)

This function allows user to download continuous waveform data using the Obspy get_waveforms function.

Parameters: wsp (*str*): FDSN Web service request client (https://docs.obspy.org/packages/autogen/obspy.clients.fdsn.client.Client.html)

net (*str*): the network code for the seismic network from which the waveform is to be downloaded

sta (*str*): the station code for the seismic station from which the waveform is to be downloaded

loc (*str*): location identifier for downloading the data

chan (*str*): channel code for downloading data

starttime (*UTCDateTime*): starting time for the downloaded data

endtime (*UTCDateTime*): ending time for the downloaded data

Returns: stream: downloaded Obspy Stream

saipy.data.realdata.preprocessing(stream, resample_freq=100, freq_min=1, freq_max=45)

This function is used to pre-process downloaded data; pre-processing steps include resampling, detrending and bandpass filtering

Parameters: stream (*obspy Stream*): stream object for preprocessing

resample_freq (*float, optional*): frequency (Hz) at which the stream is to be resampled

freq_min (*float, optional*): lower frequency (Hz) limit for bandpass filtering

freq_max (*float, optional*): upper frequency (Hz) limit for bandpass filtering

Returns: resample_stream: resampled and detrended stream object

X: bandpass filtered data in the form of numpy array

saipy.models.creime.CREIME

A CREIME object.

Attributes: model: keras Model trained as described in https://doi.org/10.1029/2022JB024595

saipy.models.creime.CREIME.get_model(untrained = False)

Parameters: untrained (*bool, optional*): if True, returns untrained model

Returns: model: keras Model for CREIME

saipy.models.creime.CREIME.predict(X)

Parameters: X: numpy array of shape 512 × 3 representing 3-component seismograms on which CREIME is to be applied

Returns: y_pred: numpy array of corresponding predicted labels in the shape 512 × 1

predictions: list of tuples with three elements corresponding to noise-vs-event classification (0 for noise and 1 for event), predicted magnitude (for events) and predicted P-arrival sample (for events)

saipy.models.creime.CREIME_RT

A CREIME_RT object, which is an improved version of CREIME capable of handling data in real time.

Attributes: model: keras Model for CREIME_RT

saipy.models.creime.CREIME_RT.get_model(untrained = False)

Parameters: untrained (*bool, optional*): if True, returns untrained model

Returns: model: keras Model for CREIME_RT

saipy.models.creime.CREIME_RT.predict(X)

Parameters: X: list of 3-component seismograms of length 6000 samples or less on which CREIME_RT is to be applied

Returns: y_pred: numpy array of corresponding predicted labels in the shape 6000 × 1

predictions: list of tuples with two elements corresponding to noise-vs-event classification (0 for noise and 1 for event) and predicted magnitude (for events)

saipy.models.polarcap.PolarCAP

A PolarCAP object.

Attributes: model: keras Model trained as described in https://doi.org/10.1016/j.aiig.2022.08.001

saipy.models.polarcap.PolarCAP. get_model(untrained=False)

Parameters: untrained (*bool, optional*): if True, returns untrained model

Returns: model: keras Model for PolarCAP

saipy.models.polarcap.PolarCAP. predict(X)

Parameters: X: numpy array of shape 64 × 1 consisting of vertical component of seismograms centered around the P-arrival sample

Returns: predictions: list of tuples of two elements representing the polarity and the probability of the prediction

DynaPicker

saipy.models.dynapicker.load_model(path)

Loading pretrained DynaPicker_v2 model.

Parameters: path (*str*): path of the entire pretrained PyTorch model e.g., './checkpoint.pt'

saipy.models.dynapicker.arguments()

parameter setting up for DynaPicker_v2 training.

Parameters: batch_size (*int*): the hyperparameter that defines the number of samples used in each iteration

lr (*float*): learning rate

epochs (*int*): epoch number

num_classes (*int*): classes numbers

patience (*int*): how many epochs to wait after last time validation loss improved

verbose (*boolean*): if True, prints a message for each validation loss improvement when using early stopping

model_save_path (*str*): the path to save trained model

saipy.models.set_seed(seed)

This function is used to ensure that results are reproducible.

Parameters: seed (*int*): seed value

Modules

saipy.modules.phaseclassification.CustomDataset(dataset)

This function allows user to create phase classification dataset in torch format.

Parameters: dataset (*numpy.array*): dataset including data and label

saipy.modules.phaseclassification.train(args, device, Train_Loader, Valid_Loader, criterion, optimizer, scheduler=None)

This function allows user to train DynaPicker_v2 on the created dataset with using early stopping.

Parameters: args: argparse module's support for command-line interfaces

device (*torch.device or int*): selected device for running PyTorch model on gpu/cpu

Train_Loader (*torch.DataLoader*): training PyTorch DataLoader instance

Valid_Loader (*torch.DataLoader*): validation PyTorch DataLoader instance

criterion: PyTorch loss function that determine the model'performance by comparing predictions with ground truth

optimizer: PyTorch optimizer that adjusting model parameters to reduce model error in each training step

scheduler: PyTorch learning rate scheduler that adjust the learning rate based on the number of epochs, default is None

saipy.modules.phaseclassification.test(args, device, model, Test_Loader, criterion)

This function allows user to test DynaPicker_v2 on the created dataset.

Parameters: args (*function*): argparse module's support for command-line interfaces

device (*torch.device or int*): selected device for running PyTorch model on gpu/cpu

model: pre-trained PyTorch model

Test_Loader (*torch.DataLoader*): Testing PyTorch DataLoader instance

criterion: PyTorch loss function that determine the model'performance by comparing predictions with ground truth

saipy.modules.pytorchtools.EarlyStopping(patience=5, verbose=False, delta=0, path='', trace_func=print)

A tool to use early stopping to deal with overfitting in PyTorch. Code source from https://github.com/Bjarten/early-stopping-pytorch.

Parameters: patience (*int*): how long to wait after last time validation loss improved

verbose (*boolean*): If True, prints a message for each validation loss improvement.

delta (*float*): minimum change in the monitored quantity to quality as an improvement

path (*str*): path for the checkpoint to be saved to

trace_func (*function*): trace print function

Packagetools

saipy.utils.packagetools.monitor1(wsp, network, station, location, channel, start_time, end_time, device, leng_win, detection_windows = 5, shift=10, picker_num_shift=1, save_result=False, path='./', file_name=None)

This function allows user to download and monitor continuous waveform data.

Parameters:

wsp (*str*): FDSN Web service request client (https://docs.obspy.org/packages/autogen/obspy.clients.fdsn.client.Client.html)

network (*str*): the network code for the seismic network from which the waveform is to be downloaded

station (*str*): the station code for the seismic station from which the waveform is to be downloaded

location (*str*): location identifier for downloading the data

channel (*str*): channel code for downloading data

start_time (*UTCDateTime*): starting time for the downloaded data

end_time (*UTCDateTime*): ending time for the downloaded data

For more details on above parameters refer to https://docs.obspy.org/packages/autogen/obspy.clients.fdsn.client.Client.get_waveforms.html

device (*torch.device or int*): selected device for running PyTorch model

leng_win (*int*): length of window used for picking body-wave arrival times

detection_windows (*int*): number of windows that should have a positive prediction made by CREIME_RT before an event is considered detected

shift (*int, optional*): shift between subsequent time windows fed to CREIME_RT for event detection

picker_num_shift (*int, optional*): shift between subsequent time windows for which DynaPicker_v2 calculated picking probabilities

save_result (*bool, optional*): if True, the results of the monitoring process are saved in a .csv file

path (*str, optional*): path to directory where csv file is to be saved

file_name (*str*): name of csv file where the results are saved, *must* be provided if save_result is set to True

Returns:

outputs: A dictionary with P-picks, S-picks, magnitudes and first-motion polarities corresponding to detected events

saipy.utils.packagetools.monitor2(path_to_stream, device, leng_win, format=None, shift = 10, picker_num_shift = 10, detection_windows = 5, save_result = False, path = './', file_name = None)

This function allows user to download and monitor continuous waveform data.

Parameters:

path_to_stream (*str*): the path to the downloaded data file, to be read as a stream using obspy.core.stream.read

device (*torch.device or int*): selected device for running PyTorch model

leng_win (*int*): length of window used for picking body-wave arrival times

format (*str, optional*): format of the file being read

shift (*int, optional*): shift between subsequent time windows fed to CREIME_RT for event detection

picker_num_shift (*int, optional*): shift between subsequent time windows for which DynaPicker_v2 calculated picking probabilities

detection_windows (*int*): number of windows that should have a positive prediction made by CREIME_RT before an event is considered detected

save_result (*bool, optional*): if True, the results of the monitoring process are saved in a .csv file

path (*str, optional*): path to directory where csv file is to be saved

file_name (*str*): name of csv file where the results are saved, *must* be provided if save_result is set to True

Returns:

outputs: A dictionary with P-picks, S-picks, magnitudes and first-motion polarities corresponding to detected events

saipy.utils.packagetools.classic_picking(trigger_type, trace, nsta, nlta, thr_on, thr_off, plotFlag)

This function allows classic event detection using short time average/long time average (sta/lta) algorithm.

Parameters:

trigger_type (*str*): this should be either 'classic_sta_lta' or 'recursive_sta_lta' which imports the corresponding function from obspy.signal.trigger

trace (*obspy trace*): this is the trace on which sta/lta is to be applied

nsta (*int*): length of window (in samples) for computing short time average

nlta (*int*): length of window (in samples) for computing long time average

thr_on (*float*): threshold at which trigger is switched on

thr_off (*float*): threshold at which trigger is switched off

plotFlag (*bool*): if True the characteristic function of the trigger is plotted on the waveform using the obspy.signal.trigger.plot_trigger function

Picktools

saipy.utils.picktools.make_stream_stead(dataset)

This function allows user to convert STEAD dataset from hdf5 to obspy stream (code source from https://github.com/smousavi05/STEAD)

Parameters: dataset (*hdf5*): hdf5 dataset

saipy.utils.picktools.make_stream_instance(df, h5, line, wftype)

This function allows user to convert INSTANCE dataset from hdf5 to obspy stream (code source from https://github.com/INGV/instance)

Parameters: df (*pandas DataFrame*): metadata of INSTANCE dataset

line (*int*): line number of the hdf5 File

wftype (*str*): waveform type 'ev_c', 'ev_gm' or 'noise' for events in counts,

events in ground motion units or noise, respectively

saipy.utils.picktools.phase_picking(device, model, st, bandpass_filter_flag, picker_num_shift , batch_size, fremin, fremax, fo, fs)

This function allows user to apply DynaPicker_v2 for phase picking on continuous waveform.

Parameters: device (*torch.device or int*): selected device for running PyTorch model

model : PyTorch model for DynaPicker_v2

stream (*obspy stream*): obspy stream format of the data used for phase picking

band_passfilter_flag (*boolean*): if Ture, apple band_pass filter

picker_num_shift (*int, optional*): shift between subsequent time windows for which DynaPicker_v2 calculated picking probabilities

batch_size (*int, optional*): the hyperparameter that defines the number of samples used in each iteration for phase arrival time estimation

fremin (*int)*: pass band low corner frequency.

fremax (*int)*: pass band high corner frequency.

fo (*int*): filter order

fs (*int*): sampling rate n Hz, default value is 100

Visualizations

A set of tools for visualizing waveform data and model outputs.

saipy.utils.visualizations.plot_waveform(data, times=None, P_arr=None, S_arr=None, magnitude=None)

Tool for visualizing 3 component seismic waveforms.

Parameters: data (*numpy array*): 3-component waveform in the form of numpy array

times (*list or numpy array, optional*): a list of times (in samples) corresponding to the data provided

P_arr (*int*): P-arrival sample

S_arr (*int*): S-arrival sample

magnitude (*float*): magnitude of the event

saipy.utils.visualizations.plot_creime_data(X,y, y_pred=None)

This is a visualization tool for data, labels and outputs corresponding to CREIME.

Parameters: X (*numpy array*): numpy array of shape 512 × 3 representing 3-component seismograms

y (*numpy array*): numpy array of shape 512 × 1 representing corresponding CREIME label

y_pred (*numpy array, optional*): numpy array of shape 512 × 1 representing corresponding CREIME prediction

saipy.utils.visualizations.plot_creime_rt_data(X,y, y_pred=None)

This is a visualization tool for data, labels and outputs corresponding to CREIME_RT.

Parameters: X (*numpy array*): numpy array of variable length representing 3-component seismograms

y (*numpy array*): numpy array of shape 6000 × 1 representing corresponding CREIME_RT label

y_pred (*numpy array, optional*): numpy array of shape 6000 × 1 representing corresponding CREIME prediction

saipy.utils.visualizations.plot_polarcap_data(X, y_true=None, y_pred=None)

This is a visualization tool for data, labels and outputs corresponding to PolarCAP.

Parameters: X (*numpy array*): numpy array of shape 64 × 1 representing the vertical component of seismograms centered around the P-arrival sample

y_true (*str, optional*): the 'known' first-motion polarity

y_true (*tuple or list, optional*): the predicted first-motion polarity and corresponding probability

saipy.utils.visualizations.plot_dynapicker_stead(stream, dataset, prob_p, prob_s, picker_num_shift, figure_size, index)

This is a visualization tool for plotting STEAD data and outputs corresponding to DynaPicker_v2.

Parameters: stream (*obspy.core.stream.Stream*): obspy stream

dataset (*hdf5 dataset*): STEAD dataset

prob_p (*float*): probability of the estimated P-phase

prob_s (*float*): probability of the estimated S-phase

picker_num_shift (*int, optional*): shift between subsequent time windows for which DynaPicker_v2 calculated picking probabilities

figure_size (*tuple*): figure size

index (*int*): index of the seismic waveform component, e.g., 0 - 'E-W', 1 - 'N-S' or 2 - 'Vertical'

saipy.utils.visualizations.plot_dynapicker_instance(stream, row, prob_p, prob_s, picker_num_shift, index, figure_size)

This is a visualization tool for INSTANCE data and outputs corresponding to DynaPicker_v2.

Parameters: stream (*obspy.core.stream.Stream*): obspy stream

row: INSTANCE dataset

prob_p (*float*): probability of the estimated P-phase

prob_s (*float*): probability of the estimated S-phase

picker_num_shift (*int, optional*): shift between subsequent time windows for which DynaPicker_v2 calculated picking probabilities

figure_size (*tuple*): figure size

saipy.utils.visualizations.plot_dynapicker_stream(stream, prob_p, prob_s, picker_num_shift, figure_size)

This is a visualization tool for stream data and outputs corresponding to DynaPicker_v2.

Parameters: stream (*obspy.core.stream.Stream*): stream data

prob_p (*float*): probability of the estimated P-phase

prob_s (*float*): probability of the estimated S-phase

picker_num_shift (*int, optional*): shift between subsequent time windows for which DynaPicker_v2 calculated picking probabilities

figure_size (*tuple*): figure size

saipy.utils.visualizations.plot_dynapicker_train_history(train_loss, valid_loss, figure_size)

This is a visualization tool for plotting DynaPicker_v2 training history

Parameters: train_loss(*list*): loss of training dataset per epoch

valid_loss(*list*): loss of validation dataset per epoch

figure_size (*tuple*): figure size

saipy.utils.visualizations.plot_dynapicker_confusionmatrix(y_true, y_pred, label_list,figure_size, cmap)

This is a visualization tool for plotting confusion matrix using DynaPicker_v2 for phase classification.

Parameters: y_true (*list*): ground truth labels

y_pred (*list*): predicted labels

label_list (*list*): list of class label like ['P-phase', 'S-phase', 'Noise']

figure_size (*tuple*): figure size in inches

cmap: colormaps in Matplotlib

saipy.utils.visualizations.plot_precision_recall_curve(y_true, y_pred, y_pred_prob, label_list, figure_size)

This is a visualization tool for plotting confusion matrix using DynaPicker_v2 for phase classification.

Parameters: y_true (*list*): ground truth labels

y_pred (*list*): predicted labels

y_pred_prob (*list*): probabilities for each data

label_list (*list*): list of class label like ['P-phase', 'S-phase', 'Noise']

figure_size (*tuple*): figure size in inches

saipy.utils.visualizations.plot_roc_curve(y_true, y_pred, y_pred_prob, label_list, figure_size)

This is a visualization tool for plotting ROC curve..

Parameters: y_true (*list*): ground truth labels

y_pred (*list*): predicted labels

y_pred_prob (*list*): probabilities for each data

label_list (*list*): list of class label like ['P-phase', 'S-phase', 'Noise']

figure_size (*tuple*): figure size in inches