

Module **bstpp.main**

Functions

```
def load_Boko_Haram()
```

Load Boko Haram dataset Returns

dict

events: event dataset from <https://ucdp.uu.se/downloads/>
(<https://ucdp.uu.se/downloads/>) covariates: covariates from PRIO-GRID
(<https://grid.prio.org/#/>) (<https://grid.prio.org/#/>)

```
def load_Chicago_Shootings()
```

Load Chicago Shootings dataset Returns

dict

Shooting report data from: <https://data.cityofchicago.org/Public-Safety/Chicago-Shootings/fsku-dr7m> (<https://data.cityofchicago.org/Public-Safety/Chicago-Shootings/fsku-dr7m>) Community Area boundaries from: <https://data.cityofchicago.org/Facilities-Geographic-Boundaries/Boundaries-Community-Areas-current-/cauq-8yn6> (<https://data.cityofchicago.org/Facilities-Geographic-Boundaries/Boundaries-Community-Areas-current-/cauq-8yn6>) Community Area Covariates from: <https://datahub.cmap.illinois.gov/maps/2a0b0316dc2c4ecfa40a171c635503f8/about> (<https://datahub.cmap.illinois.gov/maps/2a0b0316dc2c4ecfa40a171c635503f8/about>)

Classes

```
class Hawkes_Model (data, A, T, cox_background='cox',  
                    temporal_trig=bstpp.trigger.Temporal_Exponential,  
                    spatial_trig=bstpp.trigger.Spatial_Symmetric_Gaussian, **kwargs)
```

Spatiotemporal Point Process Model given by,

$$\lambda(t, s) = \mu(s, t) + \sum_{i:t_i < t} \alpha f(t - t_i; \beta) \varphi(s - s_i; \sigma^2)$$

where f is defined by spatial_trig, φ is defined by spatial_trig. If cox_background is true, μ is given by

$$\mu(s, t) = \exp(a_0 + X(s)w + f_s(s) + f_t(t))$$

where $X(s)$ is the spatial covariate matrix, f_s and f_t are Gaussian Processes. Both f_s and f_t are simulated by a pretrained VAE. We used a squared exponential kernel with hyperparameters $l \sim \text{InverseGamma}(15, 1)$ and $\sigma^2 \sim \text{LogNormal}(2, 0.5)$

Otherwise, the μ is given by

$$\mu(s, t) = \exp(a_0 + X(s)w)$$

The data is rescaled to fit in a 1x1 spatial grid and a length 50 time window. Posterior samples must be interpreted with this in mind.

Parameters

data : str or pd.DataFrame

either file path or DataFrame containing spatiotemporal data. Columns must include 'X', 'Y', 'T'.

A : np.array [2x2], GeoDataFram

Spatial region of interest. If np.array first row is the x-range, second row is y-range.

T : float

Maximum time in region of interest. Time is assumed to start at 0.

cox_background : bool

use gaussian processes in background

temporal_trig : class Trigger

an implementation of Trigger to parameterize the temporal triggering mechanism.

spatial_trig : class Trigger

an implementation of Trigger to parameterize the spatial triggering mechanism.

kwargs : dict

parameters from Point_Process_Model

Ancestors

Point_Process_Model

Methods

```
def get_params(self)
```

Returns

dict of parameter names as keys and lengths as values

```
def plot_trigger_posterior(self)
```

Plot histograms of posterior trigger parameters. Returns

pd.DataFrame

Summary of trigger parameters.

```
def plot_trigger_time_decay(self, t_units='days')
```

Plot temporal trigger kernel sample posterior.

Parameters

t_units : str

Time units of original data.

```
def simulate(self, parameters=None)
```

Simulate data from mean posterior parameters. Parameters

parameters : dict

Parameters to simulate from. If parameters is None, use mean of posterior samples. keys are string parameter names. values are np.array or float. Names must be same as those that appear in the sample from the model.

Returns

geopandas DataFrame: ['X','Y','T'] columns
simulated data

Inherited members

Point_Process_Model: cov_weight_post_summary, expected_AIC, load_rslts, log_expected_likelihood, plot_spatial, plot_temporal, run_mcmc, run_svi, save_rslts

class LGCP_Model (*args, **kwargs)

Spatiotemporal LGCP Model given by,

$$\lambda(t, s) = \exp(a_0 + X(s)w + f_s(s) + f_t(t))$$

where $X(s)$ is the spatial covariate matrix, f_s and f_t are Gaussian Processes. Both f_s and f_t are simulated by a pretrained VAE. We used a squared exponential kernel with hyperparameters $l \sim \text{InverseGamma}(15, 1)$ and $\sigma^2 \sim \text{LogNormal}(2, 0.5)$

The data is rescaled to fit in a 1x1 spatial grid and a lenght 50 time window. Posterior samples must be interpreted with this in mind.

Parameters

args : list

Parameters from Point_Process_Model

kwargs : dict

Parameters from Point_Process_Model

Ancestors

Point_Process_Model

Methods

def get_params(self)

Returns

dict of parameter names as keys and lengths as values

def simulate(self, parameters=None)

Simulate data from mean posterior parameters. Requires model inference.

Parameters

parameters : dict

Parameters to simulate from. If parameters is None, use mean of posterior samples. keys are string parameter names. values are np.array or float. Names must be same as those that appear in the sample from the model.

Returns

geopandas DataFrame: ['X', 'Y', 'T'] columns
simulated data

Inherited members

Point_Process_Model: cov_weight_post_summary, expected_AIC, load_rslts, log_expected_likelihood, plot_spatial, plot_temporal, run_mcmc, run_svi, save_rslts

```
class Point_Process_Model (model, data, A, T, spatial_cov=None, cov_names=None,
                           cov_grid_size=None, standardize_cov=True, **priors)
```

Spatiotemporal Point Process Model. The data is rescaled to fit in a 1x1 spatial grid and a length 50 time window. Posterior samples must be interpreted with this in mind.

Parameters

model : str

one of ['cox_hawkes', 'lgcp', 'hawkes'].

data : str or pd.DataFrame

either file path or DataFrame containing spatiotemporal data. Columns must include 'X', 'Y', 'T'.

A : np.array [2x2], GeoDataFram

Spatial region of interest. If np.array first row is the x-range, second row is y-range.

T : float

Maximum time in region of interest. Time is assumed to spart at 0.

spatial_cov : str, pd.DataFrame, gpd.GeoDataFrame

Either file path (.csv or .shp), DataFrame, or GeoDataFrame containing spatial covariates. Spatial covariates must cover all the points in data. If spatial_cov is a csv or pd.DataFrame, the first 2 columns must be 'X', 'Y' and cov_grid_size must be specified.

cov_names : list

List of covariate names. Must all be columns in spatial_cov.

cov_grid_size : list-like

Spatial covariate grid (width, height).

standardize_cov : bool

Standardize covariates

priors : dict

priors for parameters (a_0,w,alpha,beta,sigma_2). Must be a numpyro distribution.

Subclasses

Hawkes_Model, LGCP_Model

Methods

def cov_weight_post_summary(self)

Plot and summarize posteriors of weights and bias. Returns

pd.DataFrame

summary of weights and bias

def expected_AIC(self)

Calculate the expected AIC over the posterior distribution. For k = number of model parameters, expected AIC is defined as,

$$E_{\theta|X}[AIC] = \frac{-2}{S} \sum_{s=1} S \log(p(X|\theta^s)) + 2k$$

def load_rslts(self, file_name)

Load previously computed results Parameters

file_name : string

File where pickled results are held

def log_expected_likelihood(self, data)

Computes the log expected likelihood for test data.

$$E_{\theta^s|X}[\ell] = \log\left(\frac{1}{S} \sum_{s=1} S p(X|\theta^s)\right)$$

Parameters

data : pd.DataFrame or str

test events in the same format as original event dataset.

def plot_spatial(self, include_cov=False, **kwargs)

Plot mean posterior spatial intensity (ignoring self-excitation) with/without covariates

Parameters

include_cov : bool

Include effects of spatial covariates.

kwargs : dict

Plotting parameters for geopandas plot.

def plot_temporal(self, rescale=True)

Plot mean posterior temporal gaussian process.

Parameters

rescale : bool

Scale posteriors to original dimensions of the data.

```
def run_mcmc(self, batch_size=1, num_warmup=500, num_samples=1000, num_chains=1,
            thinning=1)
```

Run MCMC posterior sampling on model.

Parameters

batch_size : int

See numpyro documentation for description

num_warmup : int

num_samples : int

num_chains : int

thinning : int

```
def run_svi(self, num_steps, lr, num_samples=1000, resume=False, plot_loss=True,
            **kwargs)
```

Perform Stochastic Variational Inference on the model. Parameters

num_samples : int, default= 1000

Number of samples to generate after SVI.

resume : bool, default= False

Pick up where last SVI run was left off. Can only be true if model has previous run_svi call.

lr : float, default= 0.001

learning rate for SVI

num_steps : int , default= 10000
Number of iterations for SVI to run.

plot_loss : bool

auto_guide : numpyro AutoGuide , default= AutoMultivariateNormal
See numpyro AutoGuides for details.

init_strategy : function , default= init_to_median
See numpyro init strategy documentation

def save_rslts(self, file_name)

Save previously computed results Parameters

file_name : string
File where to save results

Module **bstpp.trigger**

Classes

class `Spatial_Symmetric_Gaussian (prior)`

Single parameter symmetric spatial gaussian trigger given by,

$$\varphi(\mathbf{x}; \sigma_x^2) = \frac{1}{2\pi\sigma_x} \exp\left(-\frac{1}{2\sigma_x^2} \mathbf{x} \cdot \mathbf{x}\right)$$

Abstract Trigger class to be extended for Hawkes models. The trigger is assumed to be a pdf and the reproduction rate is coded separately. The required methods to implement are:

- `compute_trigger` : compute the trigger function (pdf)
- `compute_integral` : compute the integral of the trigger function given limits (cdf)
- `get_par_names` : returns a list of the parameter names used in the trigger function

`simulate_trigger` is used only if a user wishes to simulate from the trigger function.

Parameters

prior : dict of numpyro distributions

Used to sample parameters for trigger

Ancestors

Trigger, `abc.ABC`

Inherited members

Trigger: `compute_integral`, `compute_trigger`, `get_par_names`,
`sample_parameters`, `simulate_trigger`

`class Temporal_Exponential (prior)`

Temporal exponential trigger function given by,

$$f(t; \beta) = \frac{1}{\beta} e^{-t/\beta}$$

Abstract Trigger class to be extended for Hawkes models. The trigger is assumed to be a pdf and the reproduction rate is coded separately. The required methods to implement are:

- `compute_trigger` : compute the trigger function (pdf)
- `compute_integral` : compute the integral of the trigger function given limits (cdf)
- `get_par_names` : returns a list of the parameter names used in the trigger function

`simulate_trigger` is used only if a user wishes to simulate from the trigger function.

Parameters

prior : dict of numpyro distributions

Used to sample parameters for trigger

Ancestors

Trigger, abc.ABC

Inherited members

Trigger: `compute_integral`, `compute_trigger`, `get_par_names`,
`sample_parameters`, `simulate_trigger`

`class Temporal_Power_Law (prior)`

Helper class that provides a standard way to create an ABC using inheritance.

Power Law Temporal trigger. Lomax distribution given by,

$$f(t; \beta, \gamma) = \beta \gamma^\beta (\gamma + t)^{-\beta-1}$$

Ancestors

Trigger, abc.ABC

Inherited members

Trigger: compute_integral, compute_trigger, get_par_names, sample_parameters, simulate_trigger

```
class Trigger (prior)
```

Helper class that provides a standard way to create an ABC using inheritance.

Abstract Trigger class to be extended for Hawkes models. The trigger is assumed to be a pdf and the reproduction rate is coded separately. The required methods to implement are:

- `compute_trigger` : compute the trigger function (pdf)
- `compute_integral` : compute the integral of the trigger function given limits (cdf)
- `get_par_names` : returns a list of the parameter names used in the trigger function

`simulate_trigger` is used only if a user wishes to simulate from the trigger function.

Parameters

prior : dict of numpyro distributions

Used to sample parameters for trigger

Ancestors

abc.ABC

Subclasses

Spatial_Symmetric_Gaussian, Temporal_Exponential, Temporal_Power_Law

Methods

```
def compute_integral(self, pars, limits)
```

Compute the integral of the trigger function from the given limits. For temporal triggers, the integral is computed from 0 to the upper bound. For spatial triggers, the integral is over the rectangle defined by $[[x_{\max}, x_{\min}], [y_{\max}, y_{\min}]]$ Parameters

pars : dict

results from sample_parameters

limits : jax numpy matrix

limits of integration with shape temporal - [n] compute integral from 0 to limit
spatial - [2, 2, n] compute integral over rectangle defined by $[[x_{\max}, x_{\min}], [y_{\max}, y_{\min}]]$
spatiotemporal - ([n], [2, 2, n]) combination of temporal limits and spatial limits

Returns

jax numpy [n]

def compute_trigger(self, pars, mat)

Compute the trigger function. Computes the trigger function for the [n,n] difference matrix of points. Parameters

pars : dict

results from sample_parameters

mat : jax numpy matrix

Difference matrix, whose shape is different for each kind of trigger. temporal triggers - [n, n] spatial triggers - [2, n, n] spatiotemporal triggers - [3, n, n]

Returns

jax numpy matrix [n,n]. Trigger function computed for each entry in the matrix

def get_par_names(self)

Get list of parameter names. Parameter names may not overlap with any other parameter in the model. Excluded names include ['alpha','a_0','b_0','f_xy','v_xy','f_t','v_t','w','mu_xyt','rate_t','z_spatial','z_temporal','rate_xy']. Each parameter named here must have a prior with the same name specified in the model. Returns

list of names of parameters

def sample_parameters(self)

Sample parameters using numpyro e.g. return {'beta': numpyro.sample('beta', self.prior['beta'])}

Returns

dict of a single sample of parameters

def simulate_trigger(self, pars)

Simulate a point from the trigger function (assuming the trigger is a pdf). Optional. Only necessary for data simulation. Parameters

pars : dict

parameters for the trigger to generate point.

Returns

spatial triggers - np.array [2]
temporal triggers - float