
hpo Documentation

Release 0.0.1

xuedong

May 24, 2018

CONTENTS:

1	Indices and tables	23
	Python Module Index	25
	Index	27


```
class models.Model
    Bases: object

    static generate_arms (n, path, params)

    static get_search_space ()

    static run_solver (iterations, arm, data)

class params.Param (name, min_val, max_val, init_val=None, dist='uniform', scale='log',
                    log_base=2.718281828459045, interval=None)
    Bases: object

    get_dist ()

    get_max ()

    get_min ()

    get_name ()

    get_param_range (num, stochastic=False)

    get_scale ()

    get_type ()

plots.plot_all (paths, runs, classifier_name, optimizer_name, dataset_name, idx, resource_type,
               type_plot='linear', devs=False)
```

Parameters

- **paths** –
- **runs** –
- **classifier_name** –
- **optimizer_name** –
- **dataset_name** –
- **idx** –
- **resource_type** –
- **type_plot** –
- **devs** –

Returns

```
plots.plot_bo (bo_ei_history, bo_ucb_history, random, dataset_name, model, problem)
```

```
plots.plot_hct (path, runs, classifier_name, optimizer_name, dataset_name, idx)
```

Parameters

- **path** – path to which the result image is stored
- **runs** – number of runs
- **classifier_name** – name of the classifier
- **optimizer_name** – name of the optimizer
- **dataset_name** – name of the dataset
- **idx** – id of the experiment

Returns

`plots.plot_hoo` (*path, runs, classifier_name, optimizer_name, dataset_name, idx*)

Parameters

- **path** – path to which the result image is stored
- **runs** – number of runs
- **classifier_name** – name of the classifier
- **optimizer_name** – name of the optimizer
- **dataset_name** – name of the dataset
- **idx** – id of the experiment

Returns

`plots.plot_hyperband` (*path, s_max, trials, classifier_name, optimizer_name, dataset_name, idx*)

Plot test error evaluation of hyperband with different s values.

Parameters

- **path** – path to which the result image is stored
- **s_max** – maximum number of brackets
- **trials** – number of trials of one algorithm
- **classifier_name** – name of the classifier
- **optimizer_name** – name of the optimizer
- **dataset_name** – name of the dataset
- **idx** – id of the experiment

Returns

`plots.plot_hyperband_only` (*path, trials, classifier_name, optimizer_name, dataset_name, idx*)

Plot test error evaluation of hyperband.

Parameters

- **path** – path to which the result image is stored
- **trials** – number of trials of one algorithm
- **classifier_name** – name of the classifier
- **optimizer_name** – name of the optimizer
- **dataset_name** – name of the dataset
- **idx** – id of the experiment

Returns

`plots.plot_hyperloop_only` (*path, trials, classifier_name, optimizer_name, dataset_name, idx*)

Plot test error evaluation of hyperloop.

Parameters

- **path** – path to which the result image is stored
- **trials** – number of trials of one algorithm
- **classifier_name** – name of the classifier
- **optimizer_name** – name of the optimizer

- **dataset_name** – name of the dataset
- **idx** – id of the experiment

Returns

`plots.plot_random` (*path, trials, classifier_name, optimizer_name, dataset_name, idx*)

Plot test error evaluation of random search.

Parameters

- **path** –
- **trials** –
- **classifier_name** –
- **optimizer_name** –
- **dataset_name** –
- **idx** –

Returns

`plots.plot_tpe` (*path, runs, classifier_name, optimizer_name, dataset_name, idx*)

Parameters

- **path** – path to which the result image is stored
- **runs** – number of runs
- **classifier_name** – name of the classifier
- **optimizer_name** – name of the optimizer
- **dataset_name** – name of the dataset
- **idx** – id of the experiment

Returns

class `utils.Loss` (*model, x, y, method='holdout', problem='binary'*)

Bases: `object`

evaluate_loss (***param*)

`utils.build` (*csv_path, target_index, header=None*)

`utils.cum_max` (*history*)

`utils.get_budget` (*min_units, max_units, eta*)

Compute the total budget.

Parameters

- **min_units** – minimum units of resources can be allocated to one configuration
- **max_units** – maximum units of resources can be allocated to one configuration
- **eta** – elimination proportion

Returns the corresponding total budget and the number of configurations

`utils.load_data` (*dataset*)

Function that loads the data at dataset.

Parameters **dataset** – path to the dataset

Returns separated training, validation and test dataset

`utils.log_eta(x, eta)`

Compute log of x with base eta.

Parameters

- **x** – input value
- **eta** – base

Returns rounded log_eta(x)

`utils.s_to_m(start_time, current_time)`

Function that converts time in seconds to time in minutes.

Parameters

- **start_time** – starting time in seconds
- **current_time** – current time in seconds

Returns minutes

`hyperband.hyperband_finite.hyperband_finite(model, resource_type, params, min_units, max_units, runtime, director, data, rng=<mtrand.RandomState object>, eta=4.0, budget=0, n_hyperbands=1, s_run=None, doubling=False, verbose=False)`

Hyperband with finite horizon.

Parameters

- **model** – object with subroutines to generate arms and train models
- **resource_type** – type of resource to be allocated
- **params** – hyperparameter search space
- **min_units** – minimum units of resources can be allocated to one configuration
- **max_units** – maximum units of resources can be allocated to one configuration
- **runtime** – runtime patience (in min)
- **director** – path to the directory where output are stored
- **data** – dataset to use
- **rng** – random state
- **eta** – elimination proportion
- **budget** – total budget for one bracket
- **n_hyperbands** – maximum number of hyperbands to run
- **s_run** – option to repeat a specific bracket
- **doubling** – option to decide whether we want to double the per bracket budget in the outer loop
- **verbose** – verbose option

Returns None


```
hyperband.hyperband_finite.sh_finite(model, resource_type, params, n, i, eta, big_r,
                                     director, data, rng=<mtrand.RandomState object>,
                                     track_valid=array([ 1.]), track_test=array([ 1.]), ver-
                                     bose=False)
```

Successive halving.

Parameters

- **model** – model to be trained
- **resource_type** – type of resource to be allocated
- **params** – hyperparameter search space
- **n** – number of configurations in this successive halving phase
- **i** – the number of the bracket
- **eta** – elimination proportion
- **big_r** – number of resources
- **director** – where we store the results
- **data** – dataset
- **rng** – random state
- **track_valid** – initial track vector
- **track_test** – initial track vector
- **verbose** – verbose option

Returns the dictionary of arms, the stored results and the vector of test errors

```
class ho.hct.HCTree(support, support_type, father, depth, rho, nu, tvalue, tau, sigma, box)
```

Bases: `object`

```
add_children(c, dvalue)
```

```
explore(c, dvalue)
```

```
get_change_status()
```

```
reset_change_status()
```

```
sample(c, dvalue)
```

```
update(c, dvalue)
```

```
update_node(c, dvalue)
```

```
update_path(reward, c, dvalue)
```

```
class ho.hoo.HTree(support, support_type, father, depth, rho, nu, sigma, box)
```

Bases: `object`

```
add_children()
```

```
explore()
```

```
sample(alpha)
```

```
update(alpha)
```

```
update_node(alpha)
```

```
update_path(reward, alpha)
```

```
class ho.poo.PTree (support, support_type, father, depth, rhos, nu, box)
    Bases: object

    add_children ()

    explore (k)

    explore_bis (k)

    sample (alpha, k)

    sample_bis (alpha, k)

    update_all (alpha)

    update_all_bis (alpha)

    update_node (alpha, k)

    update_node_bis (alpha, k)

    update_path (reward, alpha, k)

    update_path_bis (reward, alpha, k)

class ho.utils_ho.Box (target, fmax, nsplits, sigma, support, support_type)
    Bases: object

    std_noise (sigma)
        Stochastic target with Gaussian or uniform noise.

    std_partition ()
        Standard partitioning of a black box.

ho.utils_ho.get_rhos (nsplits, rhomax, horizon)

ho.utils_ho.loss_hct (bbox: ho.utils_ho.Box, rho, nu, c, c1, delta, sigma, horizon, director,
                    keep=False)

ho.utils_ho.loss_hoo (bbox, rho, nu, alpha, sigma, horizon, update, director, keep=False)

ho.utils_ho.loss_poo (bbox, rhos, nu, alpha, horizon, epoch)

ho.utils_ho.regret_hct (bbox, rho, nu, c, c1, delta, sigma, horizon)

ho.utils_ho.regret_hoo (bbox, rho, nu, alpha, sigma, horizon, update)

ho.utils_ho.regret_poo (bbox, rhos, nu, alpha, horizon, epoch)

ho.utils_ho.std_box (target, fmax, nsplits, sigma, support, support_type)

ho.utils_ho.std_center (support, support_type)
    Pick the center of a subregion.

ho.utils_ho.std_rand (support, support_type)
    Randomly pick a point in a subregion.

ho.utils_ho.std_split (support, support_type, nsplits)
    Split a box uniformly.
```

Parameters

- **support** – vector of support in each dimension
- **support_type** – continuous or discrete
- **nsplits** – number of splits

Returns

`ho.utils_ho.std_split_rand(support, support_type, nsplits)`

Split a box randomly.

Parameters

- **support** –
- **support_type** –
- **nsplits** –

Returns

`heuristics.hyperloop.hyperloop_finite(model, resource_type, params, min_units, max_units, runtime, director, data, rng=<mtrand.RandomState object>, eta=4.0, budget=0, n_hyperloops=1, s_run=None, doubling=False, verbose=False)`

Hyperband with finite horizon.

Parameters

- **model** – object with subroutines to generate arms and train models
- **resource_type** – type of resource to be allocated
- **params** – hyperparameter search space
- **min_units** – minimum units of resources can be allocated to one configuration
- **max_units** – maximum units of resources can be allocated to one configuration
- **runtime** – runtime patience (in min)
- **director** – path to the directory where output are stored
- **data** – dataset to use
- **rng** – random state
- **eta** – elimination proportion
- **budget** – total budget for one bracket
- **n_hyperloops** – maximum number of hyperloops to run
- **s_run** – option to repeat a specific bracket
- **doubling** – option to decide whether we want to double the per bracket budget in the outer loop
- **verbose** – verbose option

Returns

None

`heuristics.ttts.ttts(model, resource_type, params, n, i, big_r, director, data, frac=0.5, dist='Bernoulli', rng=<mtrand.RandomState object>, track_valid=array([1.]), track_test=array([1.]), verbose=False)`

Top-Two Thompson Sampling.

Parameters

- **model** – model to be trained
- **resource_type** – type of resource to be allocated
- **params** – hyperparameter search space
- **n** – number of configurations in this ttts phase

- **i** – the number of the bracket
- **big_r** – number of resources
- **director** – where we store the results
- **data** – dataset
- **frac** – threshold in ttts
- **dist** – type of prior distribution
- **rng** – random state
- **track_valid** – initial track vector
- **track_test** – initial track vector
- **verbose** – verbose option

Returns the dictionary of arms, the stored results and the vector of test errors

class `classifiers.logistic.LogisticRegression` (*input_data*, *n*, *m*)

Bases: `models.Model`

static `generate_arms` (*n*, *path*, *params*, *default=False*)

Function that generates a dictionary of configurations/arms.

Parameters

- **n** – number of arms to generate
- **path** – path to which we store the results later
- **params** – hyperparameter to be optimized
- **default** – default arm option

Returns

static `get_search_space` ()

neg_log_likelihood (*y*)

Log-likelihood Loss.

Parameters **y** – correct label vector

Returns the mean of the negative log-likelihood of the prediction, we use mean instead of sum here

to make the learning rate less dependent of the size of the minibatch size

static `run_solver` (*epochs*, *arm*, *data*, *rng=None*, *classifier=None*, *track_valid=array([1.])*, *track_test=array([1.])*, *verbose=False*)

Parameters

- **epochs** – number of epochs
- **arm** – hyperparameter configuration encoded as a dictionary
- **data** – dataset to use
- **rng** – not used here
- **classifier** – initial model, set as None by default
- **track_valid** – vector where we store the best validation errors
- **track_test** – vector where we store the test errors

- **verbose** – verbose option

Returns

zero_one (y)
Zero-one Loss.

Parameters **y** – correct label vector

Returns the zero-one Loss over the size of minibatch

```
class classifiers.mlp.HiddenLayer(rng, input_data, n, m, w=None, b=None, activation=<theano.tensor.elemwise.Elemwise object>)
```

Bases: `object`

```
class classifiers.mlp.MLP(input_data, n_in, n_hidden, n_out, rng=<mtrand.RandomState object>)
```

Bases: `models.Model`

```
static generate_arms (n, path, params, default=False)  
Function that generates a dictionary of configurations/arms.
```

Parameters

- **n** – number of arms to generate
- **path** – path to which we store the results later
- **params** – hyperparameter to be optimized
- **default** – default arm option

Returns

```
static get_search_space ()  
  
static run_solver (epochs, arm, data, rng=<mtrand.RandomState object>, classifier=None, track_valid=array([ 1.]), track_test=array([ 1.]), verbose=False)
```

Parameters

- **epochs** –
- **arm** –
- **data** –
- **rng** –
- **classifier** –
- **track_valid** –
- **track_test** –
- **verbose** –

Returns

```
class classifiers.cnn.ConvolutionPoolLayer (rng, input_data, filter_shape, image_shape, pool_size)
```

Bases: `object`

```
class classifiers.ada_sklern.Ada (problem='binary', n_estimators=50, learning_rate=1)
```

Bases: `models.Model`

```
eval ()
```

```
static generate_arms (n, path, params, default=False)
```

Function that generates a dictionary of configurations/arms.

Parameters

- **n** – number of arms to generate
- **path** – path to which we store the results later
- **params** – hyperparameter to be optimized
- **default** – default arm option

Returns

```
static get_search_space ()
```

```
static run_solver (iterations, arm, data, rng=None, problem='cont', method='5fold',  
track_valid=array([ 1.]), track_test=array([ 1.]), verbose=False)
```

Parameters

- **iterations** –
- **arm** –
- **data** –
- **rng** –
- **problem** –
- **method** –
- **track_valid** –
- **track_test** –
- **verbose** –

Returns

```
class classifiers.gbm_sklearn.GBM (problem='binary', learning_rate=0.1, n_estimators=100,  
max_depth=3, min_samples_split=2, min_samples_leaf=1,  
min_weight_fraction_leaf=0.0, subsample=1.0,  
max_features=1.0)
```

Bases: [*models.Model*](#)

```
eval ()
```

```
static generate_arms (n, path, params, default=False)
```

Function that generates a dictionary of configurations/arms.

Parameters

- **n** – number of arms to generate
- **path** – path to which we store the results later
- **params** – hyperparameter to be optimized
- **default** – default arm option

Returns

```
static get_search_space ()
```

```
static run_solver (iterations, arm, data, rng=None, problem='cont', method='5fold',  
track_valid=array([ 1.]), track_test=array([ 1.]), verbose=False)
```

Parameters

- **iterations** –
- **arm** –
- **data** –
- **rng** –
- **problem** –
- **method** –
- **track_valid** –
- **track_test** –
- **verbose** –

Returns

class classifiers.knn_sklearn.**KNN** (*problem='binary', n_neighbors=5, leaf_size=30*)

Bases: [*models.Model*](#)

eval ()

static generate_arms (*n, path, params, default=False*)

Function that generates a dictionary of configurations/arms.

Parameters

- **n** – number of arms to generate
- **path** – path to which we store the results later
- **params** – hyperparameter to be optimized
- **default** – default arm option

Returns

static get_search_space ()

static run_solver (*iterations, arm, data, rng=None, problem='cont', method='5fold', track_valid=array([1.]), track_test=array([1.]), verbose=False*)

Parameters

- **iterations** –
- **arm** –
- **data** –
- **rng** –
- **problem** –
- **method** –
- **track_valid** –
- **track_test** –
- **verbose** –

Returns

```
class classifiers.mlp_sklearn.MLP (problem='binary', hidden_layer_size=100, alpha=0.001,  
                                   learning_rate_init=0.001, beta_1=0.9, beta_2=0.999)
```

Bases: *models.Model*

```
eval ()
```

```
static generate_arms (n, path, params, default=False)
```

Function that generates a dictionary of configurations/arms.

Parameters

- **n** – number of arms to generate
- **path** – path to which we store the results later
- **params** – hyperparameter to be optimized
- **default** – default arm option

Returns

```
static get_search_space ()
```

```
static run_solver (iterations, arm, data, rng=None, problem='cont', method='5fold',  
                   track_valid=array([ 1.]), track_test=array([ 1.]), verbose=False)
```

Parameters

- **iterations** –
- **arm** –
- **data** –
- **rng** –
- **problem** –
- **method** –
- **track_valid** –
- **track_test** –
- **verbose** –

Returns

```
class classifiers.rf_sklearn.RF (problem='binary', n_estimators=10, max_features=0.5,  
                                   min_samples_split=0.3, min_samples_leaf=0.2)
```

Bases: *models.Model*

```
eval ()
```

```
static generate_arms (n, path, params, default=False)
```

Function that generates a dictionary of configurations/arms.

Parameters

- **n** – number of arms to generate
- **path** – path to which we store the results later
- **params** – hyperparameter to be optimized
- **default** – default arm option

Returns

```
static get_search_space ()
```



```
static run_solver(iterations, arm, data, rng=None, problem='cont', method='5fold',
                  track_valid=array([ 1.]), track_test=array([ 1.]), verbose=False)
```

Parameters

- **iterations** –
- **arm** –
- **data** –
- **rng** –
- **problem** –
- **method** –
- **track_valid** –
- **track_test** –
- **verbose** –

Returns

```
class classifiers.svm_skllearn.SVM(problem='binary', c=0, gamma=0, kernel='rbf')
```

Bases: [models.Model](#)

```
eval()
```

```
static generate_arms(n, path, params, default=False)
```

Function that generates a dictionary of configurations/arms.

Parameters

- **n** – number of arms to generate
- **path** – path to which we store the results later
- **params** – hyperparameter to be optimized
- **default** – default arm option

Returns

```
static get_search_space()
```

```
static run_solver(iterations, arm, data, rng=None, problem='cont', method='5fold',
                  track_valid=array([ 1.]), track_test=array([ 1.]), verbose=False)
```

Parameters

- **iterations** –
- **arm** –
- **data** –
- **rng** –
- **problem** –
- **method** –
- **track_valid** –
- **track_test** –
- **verbose** –

Returns

```
class classifiers.tree_sklearn.Tree (problem='binary', max_features=0.5, max_depth=1,  
                                     min_samples_split=0.5)
```

Bases: `models.Model`

```
eval ()
```

```
static generate_arms (n, path, params, default=False)
```

Function that generates a dictionary of configurations/arms.

Parameters

- **n** – number of arms to generate
- **path** – path to which we store the results later
- **params** – hyperparameter to be optimized
- **default** – default arm option

Returns

```
static get_search_space ()
```

```
static run_solver (iterations, arm, data, rng=None, problem='cont', method='5fold',  
                  track_valid=array([ 1.]), track_test=array([ 1.]), verbose=False)
```

Parameters

- **iterations** –
- **arm** –
- **data** –
- **rng** –
- **problem** –
- **method** –
- **track_valid** –
- **track_test** –
- **verbose** –

Returns

```
class bo.surrogates.gaussian_process.GaussianProcess (covfunc, optimize=False, useg-  
                                                     rads=False, mprior=0)
```

Bases: `object`

```
fit (x, y)
```

Fits a Gaussian Process regressor.

Parameters

- **x** (*np.ndarray, shape=(nsamples, nfeatures)*) – training instances to fit the GP
- **y** (*np.ndarray, shape=(nsamples,)*) – corresponding continuous target values to x

```
get_cov_params ()
```

Current covariance function hyperparameters.

Returns the dictionary containing covariance function hyperparameters

opt_hyp (*param_key*, *param_bounds*, *grads=None*, *n_trials=5*)

Optimizes the negative marginal log-likelihood for given hyperparameters and bounds. This is an empirical Bayes approach (or Type II maximum-likelihood).

Parameters

- **param_key** (*list*) – list of hyperparameters to optimize
- **param_bounds** (*list*) – list containing tuples defining bounds for each hyperparameter to optimize over
- **grads** – gradient matrix
- **n_trials** – number of trials

param_grad (*k_param*)

Gradient over hyperparameters. It is recommended to use *self._grad* instead.

Parameters **k_param** (*dict*) – dictionary with keys being hyperparameters and values their queried values

Returns the gradient corresponding to each hyperparameters, order given by *k_param.keys()*

predict (*x_star*, *return_std=False*)

Mean and covariances for the posterior Gaussian Process.

Parameters

- **x_star** (*np.ndarray*, *shape=((nsamples, nfeatures))*) – testing instances to predict
- **return_std** (*bool*) – whether to return the standard deviation of the posterior process

Returns mean and covariance of the posterior process for testing instances

update (*x_new*, *y_new*)

Updates the internal model with *x_new* and *y_new* instances.

Parameters

- **x_new** (*np.ndarray*, *shape=((m, nfeatures))*) – new training instances to update the model with
- **y_new** (*np.ndarray*, *shape=((m,))*) – new training targets to update the model with

class `bo.acquisition.Acquisition` (*mode*, *eps=1e-06*, ***params*)

Bases: `object`

entropy (*tau*, *mean*, *std*, *sigman*)

Predictive entropy acquisition function.

Parameters

- **tau** (*float*) – best observed function evaluation
- **mean** (*float*) – point mean of the posterior process
- **std** (*float*) – point std of the posterior process
- **sigman** (*float*) – noise variance

Returns the predictive entropy

eval (*tau*, *mean*, *std*)

Evaluates selected acquisition function.

Parameters

- **tau** (*float*) – best observed function evaluation
- **mean** (*float*) – point mean of the posterior process
- **std** (*float*) – point std of the posterior process

Returns acquisition function value

expected_improvement (*tau, mean, std*)

Expected Improvement acquisition function.

Parameters

- **tau** (*float*) – best observed function evaluation
- **mean** (*float*) – point mean of the posterior process
- **std** (*float*) – point std of the posterior process

Returns the expected improvement

gpucb (*mean, std, beta*)

Upper-confidence bound acquisition function.

Parameters

- **mean** (*float*) – point mean of the posterior process
- **std** (*float*) – point std of the posterior process
- **beta** (*float*) – constant

Returns the upper-confidence bound

probability_improvement (*tau, mean, std*)

Probability of improvement acquisition function.

Parameters

- **tau** (*float*) – best observed function evaluation
- **mean** (*float*) – point mean of the posterior process
- **std** (*float*) – point std of the posterior process

Returns the probability of improvement

class `bo.bo.BO` (*surrogate, acquisition, f, parameter_dict, n_jobs=1*)

Bases: `object`

get_result ()

Prints best result in the Bayesian Optimization procedure.

Type `OrderedDict`

Returns the point yielding best evaluation in the procedure

Type `float`

Returns the best function evaluation

run (*max_iter=10, init_evals=3, resume=False*)

Runs the Bayesian Optimization procedure.

Parameters

- **max_iter** (*int*) – number of iterations to run, default is 10

- **init_evals** (*int*) – initial function evaluations before fitting a GP, default is 3
- **resume** (*bool*) – whether to resume the optimization procedure from the last evaluation, default is *False*

sample_param ()

Randomly samples parameters over bounds.

:return a random sample of specified parameters

update_gp ()

Updates the internal model with the next acquired point and its evaluation.

class `bo.covfunc.DotProd` (*sigmaf=1.0, sigman=1e-06, bounds=None, parameters={'sigman', 'sigmaf'}*)

Bases: `object`

cov (*x, x_star*)

Computes covariance function values over *x* and *x_star*.

Parameters

- **x** (*np.ndarray, shape=(n, nfeatures)*) – instances
- **x_star** (*np.ndarray, shape=(m, nfeatures)*) – instances

Returns the computed covariance matrix

grad_matrix (*x, x_star, param*)

Computes gradient matrix for instances *x, x_star* and hyperparameter *param*.

Parameters

- **x** (*np.ndarray, shape=(n, nfeatures)*) – instances
- **x_star** (*np.ndarray, shape=(m, nfeatures)*) – instances
- **param** (*str*) – parameter to compute gradient matrix for

Returns the gradient matrix for parameter *param*

class `bo.covfunc.ExpSine` (*lscale=1.0, period=1.0, bounds=None, parameters={'lscale', 'period'}*)

Bases: `object`

cov (*x, x_star*)

Computes covariance function values over *x* and *x_star*.

Parameters

- **x** (*np.ndarray, shape=(n, nfeatures)*) – instances
- **x_star** (*np.ndarray, shape=(m, nfeatures)*) – instances

Returns the computed covariance matrix

grad_matrix (*x, x_star, param*)

Computes gradient matrix for instances *x, x_star* and hyperparameter *param*.

Parameters

- **x** (*np.ndarray, shape=(n, nfeatures)*) – instances
- **param** –

Param instances

Returns the gradient matrix for parameter *param*

```
class bo.covfunc.GammaExponential(exp_gamma=1, lscale=1, sigmaf=1, sigman=1e-06,  
                                bounds=None, parameters={'exp_gamma', 'lscale',  
                                'sigman', 'sigmaf'})
```

Bases: `object`

cov (*x, x_star*)

Computes covariance function values over *x* and *x_star*.

Parameters

- **x** (*np.ndarray, shape=((n, nfeatures))*) – instances
- **x_star** (*np.ndarray, shape=((m, nfeatures))*) – instances

Returns the computed covariance matrix

grad_matrix (*x, x_star, param*)

Computes gradient matrix for instances *x, x_star* and hyperparameter *param*.

Parameters

- **x** (*np.ndarray, shape=((n, nfeatures))*) – instances
- **x_star** (*np.ndarray, shape=((m, nfeatures))*) – instances
- **param** (*str*) – parameter to compute gradient matrix for.

Returns the gradient matrix for parameter *param*

```
class bo.covfunc.Matern(v=1, lscale=1, sigmaf=1, sigman=1e-06, bounds=None, parameters={'v',  
                                'lscale', 'sigman', 'sigmaf'})
```

Bases: `object`

cov (*x, x_star*)

Computes covariance function values over *x* and *x_star*.

Parameters

- **x** (*np.ndarray, shape=((n, nfeatures))*) – instances
- **x_star** (*np.ndarray, shape=((m, nfeatures))*) – instances

Returns the computed covariance matrix

```
class bo.covfunc.Matern32(lscale=1, sigmaf=1, sigman=1e-06, bounds=None, param-  
                                eters={'sigman', 'lscale', 'sigmaf'})
```

Bases: `object`

cov (*x, x_star*)

Computes covariance function values over *x* and *x_star*.

Parameters

- **x** (*np.ndarray, shape=((n, nfeatures))*) – instances
- **x_star** (*np.ndarray, shape=((m, nfeatures))*) – instances

Returns the computed covariance matrix

grad_matrix (*x, x_star, param*)

Computes gradient matrix for instances *x, x_star* and hyperparameter *param*.

Parameters

- **x** (*np.ndarray, shape=((n, nfeatures))*) – instances
- **x_star** (*np.ndarray, shape=((m, nfeatures))*) – instances

- **param** (*str*) – parameter to compute gradient matrix for

Returns the gradient matrix for parameter *param*

```
class bo.covfunc.Matern52 (lscale=1, sigmaf=1, sigman=1e-06, bounds=None, parameters={'sigman', 'lscale', 'sigmaf'})
```

Bases: `object`

cov (*x, x_star*)

Computes covariance function values over *x* and *x_star*.

Parameters **x** (*np.ndarray, shape=(n, nfeatures)*) – instances

Param instances

Returns the computed covariance matrix

grad_matrix (*x, x_star, param*)

Computes gradient matrix for instances *x, x_star* and hyperparameter *param*.

Parameters

- **x** (*np.ndarray, shape=(n, nfeatures)*) – instances
- **x_star** (*np.ndarray, shape=(m, nfeatures)*) – instances
- **param** (*str*) – parameter to compute gradient matrix for

:return:the gradient matrix for parameter *param*

```
class bo.covfunc.RationalQuadratic (alpha=1, lscale=1, sigmaf=1, sigman=1e-06, bounds=None, parameters={'sigman', 'lscale', 'alpha', 'sigmaf'})
```

Bases: `object`

cov (*x, x_star*)

Computes covariance function values over *x* and *x_star*.

Parameters

- **x** (*np.ndarray, shape=(n, nfeatures)*) – instances
- **x_star** (*np.ndarray, shape=(m, nfeatures)*) – instances

Returns the computed covariance matrix

grad_matrix (*x, x_star, param*)

Computes gradient matrix for instances *x, x_star* and hyperparameter *param*.

Parameters

- **x** (*np.ndarray, shape=(n, nfeatures)*) – instances
- **x_star** (*np.ndarray, shape=(m, nfeatures)*) – instances
- **param** (*str*) – parameter to compute gradient matrix for

Returns the gradient matrix for parameter *param*

```
class bo.covfunc.SquaredExponential (lscale=1, sigmaf=1.0, sigman=1e-06, bounds=None, parameters={'sigman', 'lscale', 'sigmaf'})
```

Bases: `object`

cov (*x, x_star*)

Computes covariance function values over *x* and *x_star*.

Parameters

- **x** (`np.ndarray`, `shape=((n, nfeatures))`) – instances
- **x_star** (`np.ndarray`, `shape=((n, nfeatures))`) – instances

Returns the computed covariance matrix

grad_matrix (*x*, *x_star*, *param*='lscale')

Computes gradient matrix for instances *x*, *x_star* and hyperparameter *param*.

Parameters

- **x** (`np.ndarray`, `shape=((n, nfeatures))`) – instances
- **x_star** (`np.ndarray`, `shape=((m, nfeatures))`) – instances
- **param** (*str*) – parameter to compute gradient matrix for

Returns gradient matrix for parameter *param*

`bo.covfunc.kronecker_delta` (*x*, *x_star*)

Computes Kronecker delta for rows in *x* and *x_star*.

Parameters

- **x** (`np.ndarray`, `shape=((n, nfeatures))`) – instances
- **x_star** (`np.ndarray`, `shape=((m, nfeatures))`) – instances

Returns Kronecker delta between row pairs of *x* and *x_star*

`bo.covfunc.l2_norm` (*x*, *x_star*)

Wrapper function to compute the L2 norm.

Parameters

- **x** (`np.ndarray`, `shape=((n, nfeatures))`) – instances
- **x_star** (`np.ndarray`, `shape=((m, nfeatures))`) – instances

Returns pairwise Euclidean distance between row pairs of *x* and *x_star*

class `bo.logger.BColors`

Bases: `object`

BOLD = '\x1b[1m'

ENDC = '\x1b[0m'

FAIL = '\x1b[91m'

HEADER = '\x1b[95m'

OKBLUE = '\x1b[94m'

OKGREEN = '\x1b[92m'

UNDERLINE = '\x1b[4m'

WARNING = '\x1b[93m'

class `bo.logger.EventLogger` (*bo_instance*)

Bases: `object`

print_current (*bo_instance*)

print_init (*bo_instance*)

`bo.tpe_hyperopt.combine_tracks` (*trials*)

Parameters **trials** – trials object obtained from hyperopt

Returns the track vector of test errors

`bo.tpe_hyperopt.convert_params(params)`

Parameters `params` – hyperparamter dictionary as defined in params.py

Returns search space for hyperopt

`bo.utils_bo.evaluate_dataset(csv_path, target_index, problem, model, parameter_dict,
method='holdout', seed=20, max_iter=50)`

`bo.utils_bo.evaluate_random(bo_model, loss, n_eval=20)`

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

b

`bo.acquisition`, 15
`bo.bo`, 16
`bo.covfunc`, 17
`bo.logger`, 20
`bo.surrogates.gaussian_process`, 14
`bo.tpe_hyperopt`, 20
`bo.utils_bo`, 21

c

`classifiers.ada_sklern`, 9
`classifiers.cnn`, 9
`classifiers.gbm_sklern`, 10
`classifiers.knn_sklern`, 11
`classifiers.logistic`, 8
`classifiers.mlp`, 9
`classifiers.mlp_sklern`, 11
`classifiers.rf_sklern`, 12
`classifiers.svm_sklern`, 13
`classifiers.tree_sklern`, 13

h

`heuristics.hyperloop`, 7
`heuristics.ttts`, 7
`ho.hct`, 5
`ho.hoo`, 5
`ho.poo`, 5
`ho.utils_ho`, 6
`hyperband.hyperband_finite`, 4

m

`models`, 1

p

`params`, 1
`plots`, 1

u

`utils`, 3

A

Acquisition (class in `bo.acquisition`), 15
 Ada (class in `classifiers.ada_sklearn`), 9
`add_children()` (`ho.hct.HCTree` method), 5
`add_children()` (`ho.hoo.HTree` method), 5
`add_children()` (`ho.poo.PTree` method), 6

B

BColors (class in `bo.logger`), 20
 BO (class in `bo.bo`), 16
`bo.acquisition` (module), 15
`bo.bo` (module), 16
`bo.covfunc` (module), 17
`bo.logger` (module), 20
`bo.surrogates.gaussian_process` (module), 14
`bo.tpe_hyperopt` (module), 20
`bo.utils_bo` (module), 21
 BOLD (`bo.logger.BColors` attribute), 20
 Box (class in `ho.utils_ho`), 6
`build()` (in module `utils`), 3

C

`classifiers.ada_sklearn` (module), 9
`classifiers.cnn` (module), 9
`classifiers.gbm_sklearn` (module), 10
`classifiers.knn_sklearn` (module), 11
`classifiers.logistic` (module), 8
`classifiers.mlp` (module), 9
`classifiers.mlp_sklearn` (module), 11
`classifiers.rf_sklearn` (module), 12
`classifiers.svm_sklearn` (module), 13
`classifiers.tree_sklearn` (module), 13
`combine_tracks()` (in module `bo.tpe_hyperopt`), 20
`convert_params()` (in module `bo.tpe_hyperopt`), 21
 ConvolutionPoolLayer (class in `classifiers.cnn`), 9
`cov()` (`bo.covfunc.DotProd` method), 17
`cov()` (`bo.covfunc.ExpSine` method), 17
`cov()` (`bo.covfunc.GammaExponential` method), 18
`cov()` (`bo.covfunc.Matern` method), 18
`cov()` (`bo.covfunc.Matern32` method), 18
`cov()` (`bo.covfunc.Matern52` method), 19
`cov()` (`bo.covfunc.RationalQuadratic` method), 19

`cov()` (`bo.covfunc.SquaredExponential` method), 19
`cum_max()` (in module `utils`), 3

D

DotProd (class in `bo.covfunc`), 17

E

ENDC (`bo.logger.BColors` attribute), 20
`entropy()` (`bo.acquisition.Acquisition` method), 15
`eval()` (`bo.acquisition.Acquisition` method), 15
`eval()` (`classifiers.ada_sklearn.Ada` method), 9
`eval()` (`classifiers.gbm_sklearn.GBM` method), 10
`eval()` (`classifiers.knn_sklearn.KNN` method), 11
`eval()` (`classifiers.mlp_sklearn.MLP` method), 12
`eval()` (`classifiers.rf_sklearn.RF` method), 12
`eval()` (`classifiers.svm_sklearn.SVM` method), 13
`eval()` (`classifiers.tree_sklearn.Tree` method), 14
`evaluate_dataset()` (in module `bo.utils_bo`), 21
`evaluate_loss()` (`utils.Loss` method), 3
`evaluate_random()` (in module `bo.utils_bo`), 21
 EventLogger (class in `bo.logger`), 20
`expected_improvement()` (`bo.acquisition.Acquisition` method), 16
`explore()` (`ho.hct.HCTree` method), 5
`explore()` (`ho.hoo.HTree` method), 5
`explore()` (`ho.poo.PTree` method), 6
`explore_bis()` (`ho.poo.PTree` method), 6
 ExpSine (class in `bo.covfunc`), 17

F

FAIL (`bo.logger.BColors` attribute), 20
`fit()` (`bo.surrogates.gaussian_process.GaussianProcess` method), 14

G

GammaExponential (class in `bo.covfunc`), 17
 GaussianProcess (class in `bo.surrogates.gaussian_process`), 14
 GBM (class in `classifiers.gbm_sklearn`), 10
`generate_arms()` (`classifiers.ada_sklearn.Ada` static method), 9

`generate_arms()` (classifiers.gbm_sklern.GBM static method), 10

`generate_arms()` (classifiers.knn_sklern.KNN static method), 11

`generate_arms()` (classifiers.logistic.LogisticRegression static method), 8

`generate_arms()` (classifiers.mlp.MLP static method), 9

`generate_arms()` (classifiers.mlp_sklern.MLP static method), 12

`generate_arms()` (classifiers.rf_sklern.RF static method), 12

`generate_arms()` (classifiers.svm_sklern.SVM static method), 13

`generate_arms()` (classifiers.tree_sklern.Tree static method), 14

`generate_arms()` (models.Model static method), 1

`get_budget()` (in module `utils`), 3

`get_change_status()` (ho.hct.HCTree method), 5

`get_cov_params()` (bo.surrogates.gaussian_process.GaussianProcess method), 14

`get_dist()` (params.Param method), 1

`get_max()` (params.Param method), 1

`get_min()` (params.Param method), 1

`get_name()` (params.Param method), 1

`get_param_range()` (params.Param method), 1

`get_result()` (bo.bo.BO method), 16

`get_rhos()` (in module `ho.utils_ho`), 6

`get_scale()` (params.Param method), 1

`get_search_space()` (classifiers.ada_sklern.Ada static method), 10

`get_search_space()` (classifiers.gbm_sklern.GBM static method), 10

`get_search_space()` (classifiers.knn_sklern.KNN static method), 11

`get_search_space()` (classifiers.logistic.LogisticRegression static method), 8

`get_search_space()` (classifiers.mlp.MLP static method), 9

`get_search_space()` (classifiers.mlp_sklern.MLP static method), 12

`get_search_space()` (classifiers.rf_sklern.RF static method), 12

`get_search_space()` (classifiers.svm_sklern.SVM static method), 13

`get_search_space()` (classifiers.tree_sklern.Tree static method), 14

`get_search_space()` (models.Model static method), 1

`get_type()` (params.Param method), 1

`gpucb()` (bo.acquisition.Acquisition method), 16

`grad_matrix()` (bo.covfunc.DotProd method), 17

`grad_matrix()` (bo.covfunc.ExpSine method), 17

`grad_matrix()` (bo.covfunc.GammaExponential method), 18

`grad_matrix()` (bo.covfunc.Matern32 method), 18

`grad_matrix()` (bo.covfunc.Matern52 method), 19

`grad_matrix()` (bo.covfunc.RationalQuadratic method), 19

`grad_matrix()` (bo.covfunc.SquaredExponential method), 20

H

HCTree (class in `ho.hct`), 5

HEADER (bo.logger.BColors attribute), 20

heuristics.hyperloop (module), 7

heuristics.ttts (module), 7

HiddenLayer (class in `classifiers.mlp`), 9

ho.hct (module), 5

ho.hoo (module), 5

ho.poo (module), 5

ho.utils_ho (module), 6

HTree (class in `ho.hoo`), 5

hyperband.hyperband_finite (module), 4

`hyperband_finite()` (in module `hyperband.hyperband_finite`), 4

`hyperloop_finite()` (in module `heuristics.hyperloop`), 7

K

KNN (class in `classifiers.knn_sklern`), 11

`kronecker_delta()` (in module `bo.covfunc`), 20

L

`l2_norm()` (in module `bo.covfunc`), 20

`load_data()` (in module `utils`), 3

`log_eta()` (in module `utils`), 4

LogisticRegression (class in `classifiers.logistic`), 8

Loss (class in `utils`), 3

`loss_hct()` (in module `ho.utils_ho`), 6

`loss_hoo()` (in module `ho.utils_ho`), 6

`loss_poo()` (in module `ho.utils_ho`), 6

M

Matern (class in `bo.covfunc`), 18

Matern32 (class in `bo.covfunc`), 18

Matern52 (class in `bo.covfunc`), 19

MLP (class in `classifiers.mlp`), 9

MLP (class in `classifiers.mlp_sklern`), 11

Model (class in `models`), 1

models (module), 1

N

`neg_log_likelihood()` (classifiers.logistic.LogisticRegression method), 8

O

OKBLUE (bo.logger.BColors attribute), 20

OKGREEN (bo.logger.BColors attribute), 20

opt_hyp() (bo.surrogates.gaussian_process.GaussianProcess method), 14

P

Param (class in params), 1

param_grad() (bo.surrogates.gaussian_process.GaussianProcess method), 15

params (module), 1

plot_all() (in module plots), 1

plot_bo() (in module plots), 1

plot_hct() (in module plots), 1

plot_hoo() (in module plots), 1

plot_hyperband() (in module plots), 2

plot_hyperband_only() (in module plots), 2

plot_hyperloop_only() (in module plots), 2

plot_random() (in module plots), 3

plot_tpe() (in module plots), 3

plots (module), 1

predict() (bo.surrogates.gaussian_process.GaussianProcess method), 15

print_current() (bo.logger.EventLogger method), 20

print_init() (bo.logger.EventLogger method), 20

probability_improvement() (bo.acquisition.Acquisition method), 16

PTree (class in ho.poo), 5

R

RationalQuadratic (class in bo.covfunc), 19

regret_hct() (in module ho.utils_ho), 6

regret_hoo() (in module ho.utils_ho), 6

regret_poo() (in module ho.utils_ho), 6

reset_change_status() (ho.hct.HCTree method), 5

RF (class in classifiers.rf_sklearn), 12

run() (bo.bo.BO method), 16

run_solver() (classifiers.ada_sklearn.Ada static method), 10

run_solver() (classifiers.gbm_sklearn.GBM static method), 10

run_solver() (classifiers.knn_sklearn.KNN static method), 11

run_solver() (classifiers.logistic.LogisticRegression static method), 8

run_solver() (classifiers.mlp.MLP static method), 9

run_solver() (classifiers.mlp_sklearn.MLP static method), 12

run_solver() (classifiers.rf_sklearn.RF static method), 12

run_solver() (classifiers.svm_sklearn.SVM static method), 13

run_solver() (classifiers.tree_sklearn.Tree static method), 14

run_solver() (models.Model static method), 1

S

s_to_m() (in module utils), 4

sample() (ho.hct.HCTree method), 5

sample() (ho.hoo.HTree method), 5

sample() (ho.poo.PTree method), 6

sample_bis() (ho.poo.PTree method), 6

sample_param() (bo.bo.BO method), 17

sh_finite() (in module hyperband.hyperband_finite), 4

SquaredExponential (class in bo.covfunc), 19

std_box() (in module ho.utils_ho), 6

std_center() (in module ho.utils_ho), 6

std_noise() (ho.utils_ho.Box method), 6

std_partition() (ho.utils_ho.Box method), 6

std_rand() (in module ho.utils_ho), 6

std_split() (in module ho.utils_ho), 6

std_split_rand() (in module ho.utils_ho), 6

SVM (class in classifiers.svm_sklearn), 13

T

Tree (class in classifiers.tree_sklearn), 13

ttts() (in module heuristics.ttts), 7

U

UNDERLINE (bo.logger.BColors attribute), 20

update() (bo.surrogates.gaussian_process.GaussianProcess method), 15

update() (ho.hct.HCTree method), 5

update() (ho.hoo.HTree method), 5

update_all() (ho.poo.PTree method), 6

update_all_bis() (ho.poo.PTree method), 6

update_gp() (bo.bo.BO method), 17

update_node() (ho.hct.HCTree method), 5

update_node() (ho.hoo.HTree method), 5

update_node() (ho.poo.PTree method), 6

update_node_bis() (ho.poo.PTree method), 6

update_path() (ho.hct.HCTree method), 5

update_path() (ho.hoo.HTree method), 5

update_path() (ho.poo.PTree method), 6

update_path_bis() (ho.poo.PTree method), 6

utils (module), 3

W

WARNING (bo.logger.BColors attribute), 20

Z

zero_one() (classifiers.logistic.LogisticRegression method), 9