XGBoost Regression - 'real-world' example: NYC Taxi-Fare Predictor

https://www.kaggle.com/c/new-york-city-taxi-fare-prediction

```
1 from google.colab import files
2 files.upload()
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving kaggle.json to kaggle (1).json
{'kaggle.json': b'{"username":"conorc2006","key":"c5c5a6382a7d50c022aab991694fc17f"}
```

```
1 ## Ensure the kaggle.json file is present:
2 !ls -lha kaggle.json
```

```
-rw-r--r-- 1 root root 66 Jun  9 10:12 kaggle.json
```

```
1 ## Next, install the Kaggle API client:
2 !pip install -q kaggle
```

```
1 ## The Kaggle API Client expects this file to be ~/.kaggle
2 !mkdir -p ~/.kaggle
3 !cp kaggle.json ~/.kaggle/
```

```
1 ## Permissions' change
2 !chmod 600 ~/.kaggle/kaggle.json
```

```
1 !kaggle competitions download -c new-york-city-taxi-fare-prediction
```

```
Warning: Looks like you're using an outdated API Version, please consider updating (s
train.csv.zip: Skipping, found more recently modified local copy (use --force to forc
test.csv: Skipping, found more recently modified local copy (use --force to force dow
sample_submission.csv: Skipping, found more recently modified local copy (use --force
GCP-Coupons-Instructions.rtf: Skipping, found more recently modified local copy (use
```

```
1 !pip install pyGPGO
```

```
Requirement already satisfied: pyGPGO in /usr/local/lib/python3.7/dist-packages (0.4
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pyMC3 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: theano in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: arviz>=0.11.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: dill in /usr/local/lib/python3.7/dist-packages (from p
Requirement already satisfied: semver in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: typing-extensions>=3.7.4 in /usr/local/lib/python3.7/d
Requirement already satisfied: cachetools>=4.2.1 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: theano-pymc==1.1.2 in /usr/local/lib/python3.7/dist-pa
```

```
Requirement already satisfied: fastprogress>=0.2.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: matplotlib>=3.0 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: setuptools>=38.4 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: netcdf4 in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: xarray>=0.16.1 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dis
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: cftime in /usr/local/lib/python3.7/dist-packages (from
```

```
# Load some default Python modules:

import numpy as np
import scipy as sp
import pandas as pd
import matplotlib.pyplot as plt
import xgboost as xgb
import time

from matplotlib.pyplot import rc
rc('font',**{'family':'sans-serif','sans-serif':['Helvetica']})
rc('text', usetex=False)
import seaborn as sns
plt.style.use('seaborn-whitegrid')

from collections import OrderedDict
from joblib import Parallel, delayed
from numpy.linalg import slogdet, inv, cholesky, solve
from scipy.optimize import minimize
from scipy.spatial.distance import cdist
from scipy.special import gamma
from scipy.stats import norm, t
from joblib import Parallel, delayed

from pyGPGO.logger import EventLogger
from pyGPGO.GPGO import GPGO
from pyGPGO.surrogates.GaussianProcess import GaussianProcess
from pyGPGO.surrogates.tStudentProcess import tStudentProcess
from pyGPGO.surrogates.tStudentProcess import logpdf
from pyGPGO.acquisition import Acquisition
from pyGPGO.covfunc import squaredExponential
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.metrics import mean_squared_error
from xgboost import XGBRegressor
from pandas_datareader import data

import warnings
import random
warnings.filterwarnings("ignore", category=FutureWarning)
```

```
warnings.filterwarnings("ignore", category=RuntimeWarning)
warnings.filterwarnings("ignore", category=UserWarning)
```

```
# Read data in pandas dataframe:

df_train =  pd.read_csv('/content/train.csv.zip', nrows = 1_000_000, parse_dates=["pick

```

```
# List first rows:

df_train.head()
```

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitud
0	2009-06-15 17:26:21.0000001	4.5	2009-06-15 17:26:21+00:00	-73.844311	40.721319
1	2010-01-05 16:52:16.0000002	16.9	2010-01-05 16:52:16+00:00	-74.016048	40.711303
2	2011-08-18 00:35:00.00000049	5.7	2011-08-18 00:35:00+00:00	-73.982738	40.761270
3	2012-04-21 04:30:42.0000001	7.7	2012-04-21 04:30:42+00:00	-73.987130	40.733143
4	2010-03-09 07:51:00.000000135	5.3	2010-03-09 07:51:00+00:00	-73.968095	40.768008

```
# Format 'pickup_datetime' variable:

df_train['pickup_datetime'] =  pd.to_datetime(df_train['pickup_datetime'], utc=True, fo
df_train['pickup_datetime'].head()
```

```
0   2009-06-15 17:26:21+00:00
1   2010-01-05 16:52:16+00:00
2   2011-08-18 00:35:00+00:00
3   2012-04-21 04:30:42+00:00
4   2010-03-09 07:51:00+00:00
Name: pickup_datetime, dtype: datetime64[ns, UTC]
```

```
df_train.sort_values(by = 'pickup_datetime').tail() ### June 2015 the final month

```

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_lati
286276	2015-06-30 23:38:21.0000003	26.5	2015-06-30 23:38:21+00:00	-74.008385	40.71

```
# Add time variables:

df_train['hour'] = df_train['pickup_datetime'].dt.hour
df_train['weekday'] = df_train['pickup_datetime'].dt.weekday
df_train['month'] = df_train['pickup_datetime'].dt.month
df_train['year'] = df_train['pickup_datetime'].dt.year

```

785182	2015-06-30	7.5	2015-06-30	-73.959969	40.762

```
df_train = df_train.drop(['pickup_datetime','key'], axis = 1)
df_train.head()

```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitu
0	4.5	-73.844311	40.721319	-73.841610	40.7122
1	16.9	-74.016048	40.711303	-73.979268	40.7820
2	5.7	-73.982738	40.761270	-73.991242	40.7505
3	7.7	-73.987130	40.733143	-73.991567	40.7580
4	5.3	-73.968095	40.768008	-73.956655	40.7837

```
# Remove negative fares and postive outliers:

df_train = df_train[df_train.fare_amount>=0]
df_train = df_train[df_train.fare_amount<=60]
print('New size: %d' % len(df_train))
```

```
New size: 997297
```

```
# Remove missing data:

df_train = df_train.dropna(how = 'any', axis = 'rows')
print('New size: %d' % len(df_train))
```

```
New size: 997288
```

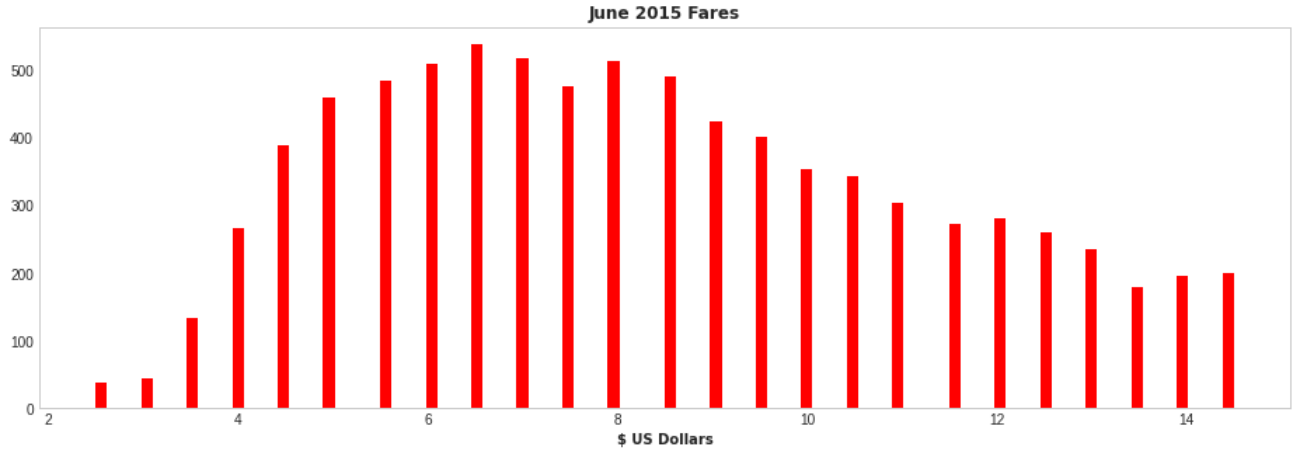
```
# June 2015 NYC taxi data (Wu et al, 2017):

df_train = df_train[df_train.month==6]
df_train = df_train[df_train.year==2015]
print('New size: %d' % len(df_train))
```

```
New size: 11269
```

```
# Histogram fare plot:
```

```
2
3 df_train[df_train.fare_amount<15].fare_amount.hist(bins=100, figsize=(16,5), color = "r
4 plt.xlabel('$ US Dollars', weight = 'bold', family = 'Arial')
5 plt.title('June 2015 Fares', weight = 'bold', family = 'Arial')
6 plt.grid(b=None)
```

```
findfont: Font family ['Arial'] not found. Falling back to DejaVu Sans.
findfont: Font family ['Arial'] not found. Falling back to DejaVu Sans.
```




```
1 y = df_train.fare_amount.values + 1e-10
2 y ### for supervised learning: output vector y
```

```
array([22.54,  8.  , 34.  , ...,  4.5 ,  6.5 ,  7.  ])
```

```
1 # List first rows (post-cleaning):
2
3 df_train.head()
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_lati
31	22.54	-74.010483	40.717667	-73.985771	40.66
310	8.00	-74.010727	40.710091	-73.998100	40.72
314	34.00	-73.974899	40.751095	-73.908546	40.88
321	8.00	-73.961784	40.759579	-73.978943	40.77
486	11.50	-73.957443	40.761703	-73.973236	40.78

```
1 X = df_train.drop(['fare_amount', 'month', 'year'], axis = 1)
2 X.head() ### for supervised learning: input matrix X
```

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passeng
31	-74.010483	40.717667	-73.985771	40.660366	
310	-74.010727	40.710091	-73.998100	40.722900	
314	-73.974899	40.751095	-73.908546	40.881878	

```
### Optimum rmse: regression model objective function is Root Mean Square Error (RMSE);
### Should be minimized (as close to zero as possible):

y_global_orig = 0
```

```
### Bayesian Optimization - inputs:

obj_func = 'XGBoost'
n_start_AcqFunc = 100
n_test = 500 # test points
df = 3 # nu

util_approx = 'ExpectedImprovement'
util_exact = 'dEI_GP'
n_init = 5 # random initialisations
opt = True

test_perc = 0.667
train_perc = 1 - test_perc

n_test = int(len(df_train) * test_perc)
n_train = int(len(df_train) - n_test)

eps = 1e-08
```

```
### Objective function:

if obj_func == 'XGBoost': # 6-D

    # Constraints:
    param_lb_alpha = 0
    param_ub_alpha = 10

    param_lb_gamma = 0
    param_ub_gamma = 10

    param_lb_max_depth = 5
    param_ub_max_depth = 15

    param_lb_min_child_weight = 1
    param_ub_min_child_weight = 20

    param_lb_subsample = .5
    param_ub_subsample = 1

    param_lb_colsample = .1
```

```
    param_ub_colsample = 1

    # 6-D inputs' parameter bounds:
    param = { 'alpha':  ('cont', (param_lb_alpha, param_ub_alpha)),
         'gamma':  ('cont', (param_lb_gamma, param_ub_gamma)),
         'max_depth':  ('int', (param_lb_max_depth, param_ub_max_depth)),
         'subsample':  ('cont', (param_lb_subsample, param_ub_subsample)),
          'min_child_weight':  ('int', (param_lb_min_child_weight, param_ub_min_child_w
            'colsample': ('cont', (param_lb_colsample, param_ub_colsample))
        }

    # True y bounds:
    dim = 6

    max_iter = 20  # iterations of Bayesian optimization

    operator = 1

    n_est = 3
```

```
### Surrogate derivatives:

cov_func = squaredExponential()

class dGaussianProcess(GaussianProcess):
    l = GaussianProcess(cov_func, optimize=opt).getcovparams()['l']
    sigmaf = GaussianProcess(cov_func, optimize=opt).getcovparams()['sigmaf']
    sigman = GaussianProcess(cov_func, optimize=opt).getcovparams()['sigman']

    def AcqGrad(self, Xstar):
        Xstar = np.atleast_2d(Xstar)
        Kstar = squaredExponential.K(self, self.X, Xstar).T
        dKstar = Kstar * cdist(self.X, Xstar).T * -1

        v = solve(self.L, Kstar.T)
        dv = solve(self.L, dKstar.T)

        ds = -2 * np.diag(np.dot(dv.T, v))
        dm = np.dot(dKstar, self.alpha)
        return ds, dm
```

```
### Set-seeds:

run_num_1 = 0
run_num_2 = 2
run_num_3 = 3
run_num_4 = 4
run_num_5 = 5
run_num_6 = 6
run_num_7 = 7
run_num_8 = 8
run_num_9 = 9
run_num_10 = 10
```

```
run_num_11 = 11
run_num_12 = 12
run_num_13 = 13
run_num_14 = 14
run_num_15 = 15
run_num_16 = 16
run_num_17 = 17
run_num_18 = 18
run_num_19 = 19
run_num_20 = 20
```

```
### Cumulative Regret Calculator:

def min_max_array(x):
    new_list = []
    for i, num in enumerate(x):
            new_list.append(np.min(x[0:i+1]))
    return new_list
```

```
class Acquisition_new(Acquisition):
    def __init__(self, mode, eps=1e-08, **params):

        self.params = params
        self.eps = eps

        mode_dict = {
            'dEI_GP': self.dEI_GP,
            'ExpectedImprovement': self.ExpectedImprovement
        }

        self.f = mode_dict[mode]

    def dEI_GP(self, tau, mean, std, ds, dm):
        gamma = (mean - tau - self.eps) / (std + self.eps)
        gamma_h = (mean - tau) / (std + self.eps)
        dsdx = ds / (2 * (std + self.eps))
        dmdx = (dm - gamma * dsdx) / (std + self.eps)

        f = (std + self.eps) * (gamma * norm.cdf(gamma) + norm.pdf(gamma))
        df1 = f / (std + self.eps) * dsdx
        df2 = (std + self.eps) * norm.cdf(gamma) * dmdx
        df = df1 + df2

        df_arr = []

        for j in range(0, dim):
          df_arr.append([df])
        return f, np.asarray(df_arr).transpose()

    def d_eval(self, tau, mean, std, ds, dm):

        return self.f(tau, mean, std, ds, dm, **self.params)
```

```
        return self.f(tau, mean, std, ds, dm, **self.params)


## dGPGO:

class dGPGO(GPGO):
    n_start = n_start_AcqFunc
    eps = 1e-08

    def d_optimizeAcq(self, method='L-BFGS-B', n_start=n_start_AcqFunc):
        start_points_dict = [self._sampleParam() for i in range(n_start)]
        start_points_arr = np.array([list(s.values())
                                     for s in start_points_dict])
        x_best = np.empty((n_start, len(self.parameter_key)))
        f_best = np.empty((n_start,))
        opt = Parallel(n_jobs=self.n_jobs)(delayed(minimize)(self.acqfunc,
                                                             x0=start_point,
                                                             method=method,
                                                             jac = True,
                                                             bounds=self.parameter_
                                           start_points_arr)
        x_best = np.array([res.x for res in opt])
        f_best = np.array([np.atleast_1d(res.fun)[0] for res in opt])

        self.x_best = x_best
        self.f_best = f_best
        self.best = x_best[np.argmin(f_best)]
        self.start_points_arr = start_points_arr

        return x_best, f_best

    def run(self, max_iter=10, init_evals=3, resume=False):

        if not resume:
            self.init_evals = init_evals
            self._firstRun(self.init_evals)
            self.logger._printInit(self)
        for iteration in range(max_iter):
            self.d_optimizeAcq()
            self.updateGP()
            self.logger._printCurrent(self)

    def acqfunc(self, xnew, n_start=n_start_AcqFunc):
        new_mean, new_var = self.GP.predict(xnew, return_std=True)
        new_std = np.sqrt(new_var + eps)
        ds, dm = self.GP.AcqGrad(xnew)
        f, df = self.A.d_eval(-self.tau, new_mean, new_std, ds=ds, dm=dm)

        return -f, df

    def acqfunc_h(self, xnew, n_start=n_start_AcqFunc, eps=eps):
        f = self.acqfunc(xnew)[0]

        new_mean_h, new_var_h = self.GP.predict(xnew + eps, return_std=True)
        new_std_h = np.sqrt(new_var_h + eps)
```

```
        ds_h, dm_h = self.GP.AcqGrad(xnew + eps)
        f_h = self.A.d_eval(-self.tau, new_mean_h, new_std_h, ds=ds_h, dm=dm_h)[0]

        approx_grad = (-f_h - f)/eps
        return approx_grad
```

```
start_approx = time.time()
start_approx
```

```
1623251773.2153277
```

```
### Bayesian optimization runs (x20): 'approx' Acquisition Function run number = 1

np.random.seed(run_num_1)
surrogate_approx_1 = dGaussianProcess(cov_func, optimize=opt)

X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=test_perc, rand

def f_syn_polarity1(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_1, obj
    score = np.array(cross_val_score(reg, X=X_train1, y=y_train1).mean())
    return operator * score

approx_1 = GPGO(surrogate_approx_1, Acquisition_new(util_approx), f_syn_polarity1, para
approx_1.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_approx_1 = approx_1.getResult()[0]
params_approx_1['max_depth'] = int(params_approx_1['max_depth'])
params_approx_1['min_child_weight'] = int(params_approx_1['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_approx_train1 = xgb.DMatrix(X_train1, y_train1)
dX_approx_test1 = xgb.DMatrix(X_test1, y_test1)
model_approx_1 = xgb.train(params_approx_1, dX_approx_train1)
pred_approx_1 = model_approx_1.predict(dX_approx_test1)

rmse_approx_1 = np.sqrt(mean_squared_error(pred_approx_1, y_test1))
rmse_approx_1
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [5.48813504 7.15189366 8.         0.92897281 8.         0.48128932].     -0
init     [ 6.45894113  4.37587211 11.          0.52835649 13.          0.44509737].
init     [ 7.91725038  5.2889492  13.          0.6963924  14.          0.40365654].
init     [ 6.48171872  3.6824154  10.          0.88907838 16.          0.88307853].
init     [4.73608045 8.00910752 8.         0.83943977 8.         0.67592892].     -0
1        [ 0.96098408  9.76459465  7.          0.75481219 17.          0.64436097].
2        [ 5.13759733  2.22657933 12.          0.58106013  2.          0.92007745].
3        [9.58067178 9.65734278 7.         0.88193436 1.         0.38223155].     -0
4        [ 0.90969339  9.80979401 14.          0.8665633   3.          0.55460209].
5        [0.3028841  4.069464   5.         0.51500749 1.         0.27359263].     -0
6        [ 8.87166351  9.3367646   5.          0.58200219 19.          0.4055524 ].
7        [ 0.51228404  8.90605177 14.          0.7949699  11.          0.73913262].
```

```
8       [ 2.05150398  0.53599727  5.          0.71551734 11.          0.37377971].
9       [8.38797278 0.6003286  6.         0.6181346  1.         0.60370114].     -0
10      [ 9.06530919  9.40796401 14.          0.73452132  4.          0.22638202].
11      [ 2.92761431  0.02841708 13.          0.97950295  9.          0.31091424].
12      [9.14092793 0.64739809 8.         0.90967035 8.         0.70435457].     -0
13      [ 1.99180311  1.76156949  6.          0.52474573 18.          0.69397695].
14      [ 1.47165443  0.32474578 13.          0.93757169 15.          0.40230457].
15      [ 3.33998723  9.61997442 13.          0.87678219 17.          0.47112812].
16      [ 0.59268574  5.88934857  8.          0.90451848 12.          0.19426621].
17      [ 7.96942817  9.76181769  7.          0.91046857 13.          0.32455735].
18      [0.45095418 9.96977628 8.         0.5330212  1.         0.30363698].     -0
19      [ 9.20494996  2.86326342  5.          0.5        20.          1.        ].
20      [3.69919707 0.85322265 6.         0.75904625 5.         0.97222422].     -0
4.667222167746296
```

```
### Bayesian optimization runs (x20): 'approx' Acquisition Function run number = 2

np.random.seed(run_num_2)
surrogate_approx_2 = dGaussianProcess(cov_func, optimize=opt)

X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, test_size=test_perc, rand

def f_syn_polarity2(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_2, obj
    score = np.array(cross_val_score(reg, X=X_train2, y=y_train2).mean())
    return operator * score

approx_2 = GPGO(surrogate_approx_2, Acquisition_new(util_approx), f_syn_polarity2, para
approx_2.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_approx_2 = approx_2.getResult()[0]
params_approx_2['max_depth'] = int(params_approx_2['max_depth'])
params_approx_2['min_child_weight'] = int(params_approx_2['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_approx_train2 = xgb.DMatrix(X_train2, y_train2)
dX_approx_test2 = xgb.DMatrix(X_test2, y_test2)
model_approx_2 = xgb.train(params_approx_2, dX_approx_train2)
pred_approx_2 = model_approx_2.predict(dX_approx_test2)

rmse_approx_2 = np.sqrt(mean_squared_error(pred_approx_2, y_test2))
rmse_approx_2
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 4.35994902  0.25926232 11.          0.97386531 12.          0.47833102].
init     [ 3.30334821  2.04648634 10.          0.55997527  6.          0.71472339].
init     [ 4.9856117   5.86796978  8.          0.89266757 11.          0.59158659].
init     [ 4.07307832  1.76984624 13.          0.75262305  7.          0.35908193].
init     [ 1.16193318  1.81727038  9.          0.79837265 19.          0.29965165].
1        [ 9.41115874  8.16019152 11.          0.70945365  2.          0.28765225].
2        [ 8.78180153  6.61060882 12.          0.91523653 18.          0.29687212].
3        [ 0.66591974  9.26661294 14.          0.96342421 18.          0.94909068].
4        [0.53023554 8.79041977 6.         0.85342606 1.         0.93968064].     -0
5        [6.47584802 1.65960359 8.         0.80737784 1.         0.90602123].     -0
```

```
6        [ 2.29390808  9.72620131 13.          0.87738596  7.          0.2041919 ].
7        [ 9.77744834  2.26597384  5.          0.98618685 19.          0.58642903].
8        [7.36877801 8.87815651 5.         0.56972286 4.         0.80625064].     -0
9        [ 5.76886466  6.30636441 11.          0.85075313  6.          0.94564556].
10       [ 0.92680624  0.80829442  5.          0.99163751 10.          0.48273491].
11       [ 8.66397635  9.94226364  5.          0.61754568 19.          0.46861962].
12       [ 2.21750241  8.1937508   6.          0.88756269 17.          0.73205426].
13       [ 9.78109337  2.52726344  5.          0.62902817 11.          0.70291169].
14       [ 2.61078484  8.48438058 14.          0.79784401 12.          0.37480602].
15       [ 9.63046012  9.46551843 14.          0.83717129 10.          0.78268661].
16       [ 9.16837918  2.32035478 14.          0.50291805  1.          0.25746282].
17       [ 1.36729794  4.89791321 13.          0.50996816  1.          0.64172429].
18       [6.0348008  1.05838904 5.         0.60659353 6.         0.66244736].     -0
19       [10.          0.66999941 12.59822409  0.5        20.          0.1       ].
20       [1.05391958 2.10485253 7.         0.55626962 1.         0.19660783].     -0
4.600295852493361
```

```
### Bayesian optimization runs (x20): 'approx' Acquisition Function run number = 3

np.random.seed(run_num_3)
surrogate_approx_3 = dGaussianProcess(cov_func, optimize=opt)

X_train3, X_test3, y_train3, y_test3 = train_test_split(X, y, test_size=test_perc, rand

def f_syn_polarity3(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_3, obj
    score = np.array(cross_val_score(reg, X=X_train3, y=y_train3).mean())
    return operator * score

approx_3 = GPGO(surrogate_approx_3, Acquisition_new(util_approx), f_syn_polarity3, para
approx_3.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_approx_3 = approx_3.getResult()[0]
params_approx_3['max_depth'] = int(params_approx_3['max_depth'])
params_approx_3['min_child_weight'] = int(params_approx_3['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_approx_train3 = xgb.DMatrix(X_train3, y_train3)
dX_approx_test3 = xgb.DMatrix(X_test3, y_test3)
model_approx_3 = xgb.train(params_approx_3, dX_approx_train3)
pred_approx_3 = model_approx_3.predict(dX_approx_test3)

rmse_approx_3 = np.sqrt(mean_squared_error(pred_approx_3, y_test3))
rmse_approx_3
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 5.50797903  7.08147823 13.          0.56066429 11.          0.11687321].
init     [ 0.40630737  2.47888297 11.          0.72040492 13.          0.23083313].
init     [ 4.53172301  2.15577008 11.          0.74631796  2.          0.60296868].
init     [ 2.59252447  4.15101197 13.          0.79330998  8.          0.24118096].
init     [ 5.44649018  7.80314765 10.          0.62879264 18.          0.44917413].
1        [1.56262424 9.7795241  5.         0.91450054 5.         0.53102391].     -0
2        [ 8.93142368  1.52910591 13.          0.84039318 17.          0.60846833].
3        [ 6.38594331  1.19109066  5.          0.81189053 13.          0.59164768].
```

```
4       [ 1.02918863  9.32189805 13.          0.88333707  1.          0.86998588].
5       [ 9.74929058  1.51205926 11.          0.50025602  8.          0.46132437].
6       [ 9.45052852  8.62641484  7.          0.79615518 14.          0.32790361].
7       [ 8.92744991  9.09956287 12.          0.74944313  3.          0.11030352].
8       [6.90239429 4.38257693 5.         0.77543741 6.         0.209803  ].     -0
9       [ 9.02893081  7.64399312 14.          0.72894099 15.          0.70609928].
10      [9.43215663 9.14652183 5.         0.95117899 1.         0.516629  ].     -0
11      [ 2.84857043  0.57472701  5.          0.53705172 19.          0.51858228].
12      [ 0.63346059  9.87550877  6.          0.74382195 14.          0.39340576].
13      [ 0.63366318  7.00411349 14.          0.89817768 18.          0.82387154].
14      [ 9.9224789   2.12085607  6.          0.82807413 19.          0.27539821].
15      [ 8.38669041  0.14009263  7.          0.51525209 10.          0.83685663].
16      [ 7.3119424   8.32659537  5.          0.53292591 18.          0.53447904].
17      [0.58114767 0.49394646 7.         0.87185676 5.         0.42563352].     -0
18      [ 2.77081064  4.33888713  7.          0.86748866 10.          0.91605352].
19      [ 9.8542409   3.34207335 14.          0.75035548  3.          0.6035725 ].
20      [4.27034104 6.89504758 8.         0.52242333 1.         0.93013039].     -0
4.649527586702978
```

```
### Bayesian optimization runs (x20): 'approx' Acquisition Function run number = 4

np.random.seed(run_num_4)
surrogate_approx_4 = dGaussianProcess(cov_func, optimize=opt)

X_train4, X_test4, y_train4, y_test4 = train_test_split(X, y, test_size=test_perc, rand

def f_syn_polarity4(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_4, obj
    score = np.array(cross_val_score(reg, X=X_train4, y=y_train4).mean())
    return operator * score

approx_4 = GPGO(surrogate_approx_4, Acquisition_new(util_approx), f_syn_polarity4, para
approx_4.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_approx_4 = approx_4.getResult()[0]
params_approx_4['max_depth'] = int(params_approx_4['max_depth'])
params_approx_4['min_child_weight'] = int(params_approx_4['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_approx_train4 = xgb.DMatrix(X_train4, y_train4)
dX_approx_test4 = xgb.DMatrix(X_test4, y_test4)
model_approx_4 = xgb.train(params_approx_4, dX_approx_train4)
pred_approx_4 = model_approx_4.predict(dX_approx_test4)

rmse_approx_4 = np.sqrt(mean_squared_error(pred_approx_4, y_test4))
rmse_approx_4
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [9.67029839 5.47232249 6.         0.92781047 9.         0.72795594].     -0
init     [ 2.16089496  9.76274455 12.          0.62649118  9.          0.66966679].
init     [ 0.05159149  5.72356491  9.          0.99170034 10.          0.10808749].
init     [ 3.86571283  0.44160058 10.          0.90553105 18.          0.95407958].
init     [ 7.86305986  8.66289299  6.          0.53285477 14.          0.25117497].
1        [ 8.45443649  8.61014312 11.          0.83475494  1.          0.14018305].
```

```
2       [ 0.77431146  1.96668116 12.         0.50723361  3.         0.74768925].
3       [2.27858743 6.23199766 5.         0.58705984 2.         0.80794289].     -0
4       [ 6.832625    9.87635293 14.         0.84450885 19.         0.20389666].
5       [ 7.37255369  2.03491596 13.         0.8921741   9.         0.46934318].
6       [ 0.05992751  6.06320143 14.         0.89322475 17.         0.32211096].
7       [ 0.79250634  6.36332745  6.         0.84703891 17.         0.8628914 ].
8       [9.26767626 0.09691703 5.         0.59554562 3.         0.95249041].     -0
9       [ 9.93824172  2.34876498  7.         0.94210707 16.         0.6947317 ].
10      [3.43076773 0.51115291 6.         0.80398076 7.         0.163561  ].     -0
11      [8.12039932 9.82838311 5.         0.6851891  3.         0.87462757].     -0
12      [ 6.03647398  5.02077859  7.         0.51964174 12.         0.90396407].
13      [ 9.03174101  2.34471053 13.         0.52042845  3.         0.20136175].
14      [ 9.29091353  0.24848851 14.         0.65742606 17.         0.33836791].
15      [3.19488299 9.73527155 6.         0.65266597 9.         0.60657999].     -0
16      [ 2.02212851  9.9407933  10.         0.71009222  3.         0.59925198].
17      [ 7.85427932  8.81949282 14.         0.936894    6.         0.6768425 ].
18      [ 6.65760625  2.35253217 11.         0.58483338 14.         0.21088543].
19      [ 0.9141999   0.89707594 12.         0.91410151 11.         0.69348379].
20      [ 6.39825002  5.88858918  9.         0.67432081 18.         0.813834  ].
5.04081354533672
```

```
1 ### Bayesian optimization runs (x20): 'approx' Acquisition Function run number = 5
2
3 np.random.seed(run_num_5)
4 surrogate_approx_5 = dGaussianProcess(cov_func, optimize=opt)
5
6 X_train5, X_test5, y_train5, y_test5 = train_test_split(X, y, test_size=test_perc, rand
7
8 def f_syn_polarity5(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
9     reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
10          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_5, obj
11    score = np.array(cross_val_score(reg, X=X_train5, y=y_train5).mean())
12    return operator * score
13
14 approx_5 = GPGO(surrogate_approx_5, Acquisition_new(util_approx), f_syn_polarity5, para
15 approx_5.run(max_iter = max_iter, init_evals = n_init) # run
16
17 ### Return optimal parameters' set:
18 params_approx_5 = approx_5.getResult()[0]
19 params_approx_5['max_depth'] = int(params_approx_5['max_depth'])
20 params_approx_5['min_child_weight'] = int(params_approx_5['min_child_weight'])
21
22 ### Re-train with optimal parameters, run predictons:
23 dX_approx_train5 = xgb.DMatrix(X_train5, y_train5)
24 dX_approx_test5 = xgb.DMatrix(X_test5, y_test5)
25 model_approx_5 = xgb.train(params_approx_5, dX_approx_train5)
26 pred_approx_5 = model_approx_5.predict(dX_approx_test5)
27
28 rmse_approx_5 = np.sqrt(mean_squared_error(pred_approx_5, y_test5))
29 rmse_approx_5
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 2.21993171  8.70732306 11.         0.68186845 10.         0.53957007].
init     [ 6.11743863  7.65907856  5.         0.64840025 16.         0.82745351].
init     [ 6.49458883  8.19472793  6.         0.93996852 19.         0.36647194].
init     [ 6.28787909  5.7983781   6.         0.63290956 17.         0.18402673].
```

```
init     [8.26554249 8.33492742 9.         0.97900675 3.         0.26957319].     -0
1        [1.95474956 1.21548467 5.         0.65548996 6.         0.3261206 ].     -0
2        [ 8.68915106  0.84881749 13.         0.9945373   7.         0.34624572].
3        [ 6.12310163  2.21013771 14.         0.74740665 18.         0.42989776].
4        [ 9.64635884  9.52633265 14.         0.67611826 10.         0.87506739].
5        [ 0.42801231  0.53056997 14.         0.88001393  3.         0.60870677].
6        [ 0.33154545  2.45627958  8.         0.87454095 14.         0.94998466].
7        [9.67414353 2.68949571 5.         0.62350468 9.         0.83649418].     -0
8        [ 0.5259471   9.7567347  14.         0.91097263  1.         0.64281486].
9        [9.64880583 0.85455793 9.         0.80366346 1.         0.54097439].     -0
10       [1.24717977 9.40645683 5.         0.65383928 3.         0.68852908].     -0
11       [ 1.7633545   9.18687735 14.         0.62483521 19.         0.67316164].
12       [7.9980796  9.13466128 6.         0.81085598 9.         0.18640376].     -0
13       [ 3.21802975  4.52251367 10.         0.91931377  1.         0.12302035].
14       [ 3.00873322  0.14737527 12.         0.50222445 10.         0.34409645].
15       [ 8.82566401  9.52943351 11.         0.94131416 17.         0.11574263].
16       [ 9.86354567  3.83813535  9.         0.75924362 13.         0.24806309].
17       [ 1.01828137  5.72294894 15.         1.         6.58067819  0.1       ].
18       [ 0.39655743  9.88537948  7.         0.8395494  15.         0.83639512].
19       [0.86243549 8.92584837 5.         0.6740487  9.         0.55305648].     -0
20       [ 9.94984248  2.37676311  9.         0.96621815 19.         0.48539202].
5.047520592626375
```

```
### Bayesian optimization runs (x20): 'approx' Acquisition Function run number = 6

np.random.seed(run_num_6)
surrogate_approx_6 = dGaussianProcess(cov_func, optimize=opt)

X_train6, X_test6, y_train6, y_test6 = train_test_split(X, y, test_size=test_perc, rand

def f_syn_polarity6(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_6, obj
    score = np.array(cross_val_score(reg, X=X_train6, y=y_train6).mean())
    return operator * score

approx_6 = GPGO(surrogate_approx_6, Acquisition_new(util_approx), f_syn_polarity6, para
approx_6.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_approx_6 = approx_6.getResult()[0]
params_approx_6['max_depth'] = int(params_approx_6['max_depth'])
params_approx_6['min_child_weight'] = int(params_approx_6['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_approx_train6 = xgb.DMatrix(X_train6, y_train6)
dX_approx_test6 = xgb.DMatrix(X_test6, y_test6)
model_approx_6 = xgb.train(params_approx_6, dX_approx_train6)
pred_approx_6 = model_approx_6.predict(dX_approx_test6)

rmse_approx_6 = np.sqrt(mean_squared_error(pred_approx_6, y_test6))
rmse_approx_6
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [8.92860151 3.31979805 5.         0.99251441 2.         0.57683563].     -0
init     [4.18807429 3.35407849 9.         0.87750649 3.         0.56623277].     -0
```

```
init     [ 5.788586    6.45355096 14.          0.70660047 12.          0.82154882].
init     [4.58184578 6.73834679 5.         0.90108528 3.         0.65482895].     -0
init     [ 4.42510505  5.75952352 14.          0.97882365 15.          0.29525604].
1        [ 2.61343239  0.80193947  5.          0.83898129 13.          0.84644718].
2        [ 9.72322443  9.21177696  5.          0.87917074 11.          0.86681736].
3        [ 5.94160899  0.75325837 13.          0.80313713  9.          0.38465378].
4        [ 8.54451719  6.55901746 13.          0.7507391   5.          0.45426156].
5        [ 9.98276075  2.51129284  8.          0.78524028 18.          0.90705498].
6        [ 1.15162056  8.90964142  5.          0.51235759 16.          0.30257234].
7        [ 0.85068891  9.82901317 11.          0.8341869   1.          0.33880369].
8        [ 9.32420466  6.39616005 13.          0.93300527 17.          0.34904443].
9        [ 8.18088231  9.44781209  7.          0.95869221 19.          0.77401414].
10       [ 0.19816257  1.95434851 14.          0.59292238  7.          0.17971262].
11       [ 0.56219975  9.42928268 11.          0.51903033 19.          0.52214278].
12       [ 1.90877122  0.88669904 11.          0.72802974 19.          0.27519153].
13       [ 0.45596011  8.24800111 12.          0.59058729  9.          0.26402873].
14       [ 9.66151696  2.12034328  6.          0.87947979 12.          0.86985697].
15       [ 3.68500147  6.04160102  8.          0.55594848 12.          0.2508545 ].
16       [0.52725495 9.82259272 6.         0.62080595 7.         0.68770749].     -0
17       [1.20916044 1.67996637 8.         0.95434584 8.         0.65713107].     -0
18       [ 8.19647777  0.42845261 12.48159385  0.5         3.75185604  1.        ].
19       [ 0.17525712  4.19183624 13.          0.93093846  1.          0.77496282].
20       [ 1.59392983  0.4187331   5.          0.53325118 19.          0.2829738 ].
4.6880443649469035
```

```
### Bayesian optimization runs (x20): 'approx' Acquisition Function run number = 7

np.random.seed(run_num_7)
surrogate_approx_7 = dGaussianProcess(cov_func, optimize=opt)

X_train7, X_test7, y_train7, y_test7 = train_test_split(X, y, test_size=test_perc, rand

def f_syn_polarity7(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_7, obj
    score = np.array(cross_val_score(reg, X=X_train7, y=y_train7).mean())
    return operator * score

approx_7 = GPGO(surrogate_approx_7, Acquisition_new(util_approx), f_syn_polarity7, para
approx_7.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_approx_7 = approx_7.getResult()[0]
params_approx_7['max_depth'] = int(params_approx_7['max_depth'])
params_approx_7['min_child_weight'] = int(params_approx_7['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_approx_train7 = xgb.DMatrix(X_train7, y_train7)
dX_approx_test7 = xgb.DMatrix(X_test7, y_test7)
model_approx_7 = xgb.train(params_approx_7, dX_approx_train7)
pred_approx_7 = model_approx_7.predict(dX_approx_test7)

rmse_approx_7 = np.sqrt(mean_squared_error(pred_approx_7, y_test7))
rmse_approx_7
```

```
    Evaluation       Proposed point           Current eval.          Best eval.
```

```
init     [0.76308289 7.79918792 8.         0.98911145 8.         0.98019056].     -0
init     [ 5.3849587   5.01120464 13.         0.74994125  5.         0.88192131].
init     [ 3.30839249  3.9294231  12.         0.6440728  13.         0.41137564].
init     [9.29528191 2.6258377  5.         0.80027446 1.         0.86616513].     -0
init     [ 1.74052764  7.90763512 14.         0.7244129   4.         0.77536887].
1        [3.43305102 3.00339076 8.         0.71322679 4.         0.33322219].     -0
2        [ 7.6343627   1.31181598  5.         0.5769645  12.         0.84874959].
3        [ 9.12127254  9.64651695 14.         0.53624962  1.         0.38247449].
4        [ 4.51243396  9.79601217  8.         0.69773915 19.         0.69687222].
5        [ 8.06748781  9.6311716   5.         0.89165168 11.         0.90769253].
6        [ 9.84853722  9.76587477 12.         0.66808012 15.         0.83895675].
7        [6.4915356  8.69600226 5.         0.66481282 2.         0.54976375].     -0
8        [ 0.86712134  2.53401598  5.         0.75823741 18.         0.7884932 ].
9        [ 1.29932493  8.0055142  14.         0.51325501 19.         0.83654101].
10       [ 8.667653    2.8973594  14.         0.53825633 18.         0.10643456].
11       [ 9.89504759  2.29066357 10.         0.72562459  8.         0.65324681].
12       [ 9.94824081  0.39149062 12.         0.59113076  1.         0.49709751].
13       [ 9.60964399  8.52338044  5.         0.92796141 17.         0.69277707].
14       [ 3.34596156  0.67230605 11.         0.7366642  19.         0.29949997].
15       [ 9.60816545  8.33912452 11.         0.58307044  8.         0.36972424].
16       [ 0.47345619  1.31956961  7.         0.68987164 11.         0.9514247 ].
17       [ 8.96081021  0.38413512 14.         0.90624871 12.         0.41560009].
18       [ 9.71411962  1.71380043  5.         0.50726337 19.         0.71982498].
19       [ 0.18764716  0.58029521 13.         0.75995865  6.         0.69134836].
20       [ 1.77048754  9.0838881   6.         0.98375355 14.         0.13025417].
4.499924380514474
```

```
### Bayesian optimization runs (x20): 'approx' Acquisition Function run number = 8

np.random.seed(run_num_8)
surrogate_approx_8 = dGaussianProcess(cov_func, optimize=opt)

X_train8, X_test8, y_train8, y_test8 = train_test_split(X, y, test_size=test_perc, rand

def f_syn_polarity8(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_8, obj
    score = np.array(cross_val_score(reg, X=X_train8, y=y_train8).mean())
    return operator * score

approx_8 = GPGO(surrogate_approx_8, Acquisition_new(util_approx), f_syn_polarity8, para
approx_8.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_approx_8 = approx_8.getResult()[0]
params_approx_8['max_depth'] = int(params_approx_8['max_depth'])
params_approx_8['min_child_weight'] = int(params_approx_8['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_approx_train8 = xgb.DMatrix(X_train8, y_train8)
dX_approx_test8 = xgb.DMatrix(X_test8, y_test8)
model_approx_8 = xgb.train(params_approx_8, dX_approx_train8)
pred_approx_8 = model_approx_8.predict(dX_approx_test8)

rmse_approx_8 = np.sqrt(mean_squared_error(pred_approx_8, y_test8))
rmse_approx_8
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 8.73429403  9.68540663 10.          0.68875849  9.          0.48011572].
init     [ 6.12033333  7.66062926  8.          0.76133734 13.          0.93379456].
init     [ 1.46524679  7.01527914  7.          0.90913299 10.          0.36016753].
init     [ 9.73855241  3.33774046 14.          0.53290419  7.          0.7088681 ].
init     [ 3.00618018  1.82702795 11.          0.75681389 14.          0.98627449].
1        [4.42022545 5.48487111 9.         0.97165909 3.         0.63617522].     -0
2        [ 9.3432851   3.80536023 13.          0.82203895 19.          0.99569116].
3        [ 2.52429836  9.02824683 14.          0.59641093 17.          0.61934886].
4        [ 9.53473907  5.08424998 11.          0.50652828 18.          0.67121466].
5        [6.89072012 1.88822945 5.         0.9252956  8.         0.40577637].     -0
6        [ 2.42575645  9.87357367  5.          0.61143882 19.          0.11833201].
7        [ 0.9931658   1.40870497 14.          0.50614818  6.          0.71944448].
8        [ 9.09148899  1.42093493  5.          0.88801001 17.          0.53231573].
9        [9.69554908 3.70013633 5.         0.72727157 1.         0.49530336].     -0
10       [ 2.66410938  9.83743919 13.          0.58899688  8.          0.29675269].
11       [1.66316873 0.47469495 7.         0.60820298 1.         0.50689458].     -0
12       [ 1.2277143   1.05411485  5.          0.50198686 13.          0.89258454].
13       [5.63420638 9.7026496  5.         0.64869539 8.         0.22475566].     -0
14       [ 0.63194856  9.08760061 12.          0.51277257  1.          0.85579144].
15       [ 8.35277365  5.17929354 14.          0.67305838  1.          0.70101595].
16       [ 8.62413803  1.63568526 10.          0.78670071 12.          0.5099252 ].
17       [ 3.5121134   5.24088616  9.          0.68973249 19.          0.78159968].
18       [5.83203328 9.4812958  5.         0.60479193 1.         0.24995758].     -0
19       [ 3.45271174  0.75931377  5.          0.81313484 19.          0.57119025].
20       [10.          0.         10.00991488  0.5         1.          0.1       ].
4.797573101898328
```

```
### Bayesian optimization runs (x20): 'approx' Acquisition Function run number = 9

np.random.seed(run_num_9)
surrogate_approx_9 = dGaussianProcess(cov_func, optimize=opt)

X_train9, X_test9, y_train9, y_test9 = train_test_split(X, y, test_size=test_perc, rand

def f_syn_polarity9(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_9, obj
    score = np.array(cross_val_score(reg, X=X_train9, y=y_train9).mean())
    return operator * score

approx_9 = GPGO(surrogate_approx_9, Acquisition_new(util_approx), f_syn_polarity9, para
approx_9.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_approx_9 = approx_9.getResult()[0]
params_approx_9['max_depth'] = int(params_approx_9['max_depth'])
params_approx_9['min_child_weight'] = int(params_approx_9['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_approx_train9 = xgb.DMatrix(X_train9, y_train9)
dX_approx_test9 = xgb.DMatrix(X_test9, y_test9)
model_approx_9 = xgb.train(params_approx_9, dX_approx_train9)
pred_approx_9 = model_approx_9.predict(dX_approx_test9)

rmse_approx_9 = np.sqrt(mean_squared_error(pred_approx_9, y_test9))
```

```
rmse_approx_9 = np.sqrt(mean_squared_error(pred_approx_9, y_test9))
rmse_approx_9
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 0.10374154  5.01874592 11.          0.50377155  2.          0.29670281].
init     [ 4.18508181  2.48101168 13.          0.69794293  2.          0.25009871].
init     [ 8.78559086  9.50964032 13.          0.98395204 11.          0.90820641].
init     [ 6.66898973  5.47837783  6.          0.97165345 12.          0.72499481].
init     [ 8.24870465  4.65668475 13.          0.68760467  9.          0.98502332].
1        [ 4.77974014  4.35555465  8.          0.84383705 19.          0.93268007].
2        [ 4.24955662  9.67331527 12.          0.64012695  7.          0.96617478].
3        [ 3.60566534  9.79805332 11.          0.62032576 16.          0.3578496 ].
4        [5.1421602  8.15576484 5.         0.73724996 3.         0.93160054].     -0
5        [ 0.42678797  0.40430921 14.          0.96202194 10.          0.10119674].
6        [ 6.47759341  0.32877119 12.          0.97031193 18.          0.58020409].
7        [9.20185355 9.40235017 8.         0.60433558 1.         0.94540222].     -0
8        [2.23847616 1.05481765 6.         0.88971719 7.         0.87464495].     -0
9        [9.98066288 1.46868871 7.         0.68668915 6.         0.1760674 ].     -0
10       [ 9.89680111  6.16985579 13.          0.58428363 17.          0.77251451].
11       [ 0.26297271  5.35234043 10.          0.52938199  9.          0.37477938].
12       [ 7.86216061  0.28362131  5.          0.73470189 17.          0.13041415].
13       [ 2.40968827  3.05909821 10.          0.83751775 14.          0.11529988].
14       [ 0.80844375  8.9121894   5.          0.56134557 18.          0.84901701].
15       [ 9.75708751  9.52851232  5.          0.99291385 16.          0.26645966].
16       [ 8.77945628  5.51694736 12.          0.84613589  4.          0.64741755].
17       [ 3.89448326  9.77793126 11.          0.87312755  1.          0.25420157].
18       [ 0.48953831  1.43059038 10.          0.65735568 19.          0.64376156].
19       [ 9.40169118 10.          7.36177871  0.60893885 10.73576509  0.6846292 ].
20       [ 2.97786479  9.66872573  7.          0.62646987 10.          0.90664407].
4.6575353004618085
```

```
### Bayesian optimization runs (x20): 'approx' Acquisition Function run number = 10

np.random.seed(run_num_10)
surrogate_approx_10 = dGaussianProcess(cov_func, optimize=opt)

X_train10, X_test10, y_train10, y_test10 = train_test_split(X, y, test_size=test_perc,

def f_syn_polarity10(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_10, ob
    score = np.array(cross_val_score(reg, X=X_train10, y=y_train10).mean())
    return operator * score

approx_10 = GPGO(surrogate_approx_10, Acquisition_new(util_approx), f_syn_polarity10, p
approx_10.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_approx_10 = approx_10.getResult()[0]
params_approx_10['max_depth'] = int(params_approx_10['max_depth'])
params_approx_10['min_child_weight'] = int(params_approx_10['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_approx_train10 = xgb.DMatrix(X_train10, y_train10)
dX_approx_test10 = xgb.DMatrix(X_test10, y_test10)
model_approx_10 = xgb.train(params_approx_10, dX_approx_train10)
pred_approx_10 = model_approx_10.predict(dX_approx_test10)
```

```
27
28 rmse_approx_10 = np.sqrt(mean_squared_error(pred_approx_10, y_test10))
29 rmse_approx_10
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 7.71320643  0.20751949  5.          0.72150747 17.          0.12265456].
init     [ 7.0920801   2.65566127 13.          0.57518893 17.          0.83494165].
init     [ 3.36071584  8.90816531  6.          0.86087766 15.          0.75469196].
init     [ 5.40880931  1.31458152  8.          0.57108502 14.          0.62551123].
init     [1.82631436 8.26082248 6.         0.80888349 5.         0.15900694].     -0
1        [8.31989768 3.09778055 7.         0.64798085 3.         0.98471878].     -0
2        [ 1.51483713  6.46720195 14.          0.87676044  8.          0.10934204].
3        [ 1.40638864  6.36994003 14.          0.94554832 18.          0.42624162].
4        [ 6.47425096  8.4482791  10.          0.69239539 11.          0.47622913].
5        [ 9.63766998  0.33667719 14.          0.56957963  7.          0.90222914].
6        [ 2.70513667  1.83577987 12.          0.56122064  1.          0.15848231].
7        [ 8.5382995   9.82970528 10.          0.72827815  4.          0.91090853].
8        [ 0.13114685  2.69967978  5.          0.89490476 19.          0.55706236].
9        [ 9.76455747  9.71578983 14.          0.66293675 18.          0.1137209 ].
10       [ 8.91866592  7.39892211  8.          0.67814307 19.          0.42475225].
11       [0.40833691 2.05040396 5.         0.83675528 2.         0.93839002].     -0
12       [ 0.06083858  0.83347778 13.          0.60208867 14.          0.81102115].
13       [ 3.87733745  1.80097513 10.          0.87276131  7.          0.20829771].
14       [ 4.19755202  9.09268434 14.          0.54945508  1.          0.70144661].
15       [9.29172085 4.53921153 5.         0.72490904 9.         0.80222442].     -0
16       [ 1.47623714  8.58014874 12.          0.98501224 13.          0.73010278].
17       [ 8.55266632  0.8729167  12.          0.54639494  1.          0.10341981].
18       [ 3.51348801  1.47885981  5.          0.61755906 10.          0.59560105].
19       [ 9.70082045  2.90861386 11.          0.53565772 13.          0.71127619].
20       [6.46081033 9.19989249 5.         0.99579878 1.         0.40331905].     -0
4.704422370494913
```

```
1 ### Bayesian optimization runs (x20): 'approx' Acquisition Function run number = 11
2
3 np.random.seed(run_num_11)
4 surrogate_approx_11 = dGaussianProcess(cov_func, optimize=opt)
5
6 X_train11, X_test11, y_train11, y_test11 = train_test_split(X, y, test_size=test_perc,
7
8 def f_syn_polarity11(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
9     reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
10          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_11, ob
11    score = np.array(cross_val_score(reg, X=X_train11, y=y_train11).mean())
12    return operator * score
13
14 approx_11 = GPGO(surrogate_approx_11, Acquisition_new(util_approx), f_syn_polarity11, p
15 approx_11.run(max_iter = max_iter, init_evals = n_init) # run
16
17 ### Return optimal parameters' set:
18 params_approx_11 = approx_11.getResult()[0]
19 params_approx_11['max_depth'] = int(params_approx_11['max_depth'])
20 params_approx_11['min_child_weight'] = int(params_approx_11['min_child_weight'])
21
22 ### Re-train with optimal parameters, run predictons:
23 dX_approx_train11 = xgb.DMatrix(X_train11, y_train11)
24 dX_approx_test11 = xgb.DMatrix(X_test11, y_test11)
```

```
model_approx_11 = xgb.train(params_approx_11, dX_approx_train11)
pred_approx_11 = model_approx_11.predict(dX_approx_test11)

rmse_approx_11 = np.sqrt(mean_squared_error(pred_approx_11, y_test11))
rmse_approx_11
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 1.80269689  0.19475241  6.         0.59705781 13.         0.47818324].
init     [ 4.85427098  0.12780815  5.         0.91309068 14.         0.86571558].
init     [ 7.2996447   1.08736072 10.         0.92857712 18.         0.66910061].
init     [ 0.20483613  1.16737269  7.         0.57895615 16.         0.83644782].
init     [ 3.44624491  3.18798797 14.         0.54197657 15.         0.63958906].
1        [9.77136617 6.6548802  7.         0.51036649 9.         0.81011527].     -0
2        [0.5279662  8.15331655 5.         0.83127487 9.         0.53242685].     -0
3        [8.62555756 1.5478147  8.         0.99964468 2.         0.74874718].     -0
4        [ 0.90299561  9.42808632 14.         0.71344248  9.         0.5250902 ].
5        [ 5.37271973  6.74878506 12.         0.69507758  3.         0.34109729].
6        [0.4982028  3.39252027 6.         0.94894432 2.         0.3491488 ].     -0
7        [ 5.0402397   9.42026535  8.         0.71577    16.         0.52045581].
8        [ 0.07362338  1.12762958 13.         0.93121884  6.         0.44481438].
9        [ 8.74582539  1.09960491 14.         0.69981761  9.         0.73083826].
10       [ 9.62795963  5.53773092  5.         0.74613073 18.         0.44845649].
11       [ 8.61765552  0.94349031 10.         0.85641829  4.         0.91576259].
12       [3.50049766 0.06230783 7.         0.82711607 1.         0.94515625].     -0
13       [5.71344477 9.31745465 5.         0.62710446 1.         0.57350664].     -0
14       [ 8.40730244  5.3235921  14.         0.57824885  9.         0.96689527].
15       [ 3.80424221  9.01046858 14.         0.56083724 18.         0.27280598].
16       [ 9.7602473   8.08368399 12.         0.65958526 18.         0.9544452 ].
17       [ 7.19988755  9.76929986 14.         0.62177947 13.         0.86328626].
18       [ 6.19006075  3.41925871 11.         0.95286331  6.         0.39133224].
19       [0.77256475 1.22732493 7.         0.97692145 7.         0.9071396 ].     -0
20       [7.46566309 0.73949694 7.         0.58224414 9.         0.25972328].     -0
5.084919441252919
```

```
### Bayesian optimization runs (x20): 'approx' Acquisition Function run number = 12

np.random.seed(run_num_12)
surrogate_approx_12 = dGaussianProcess(cov_func, optimize=opt)

X_train12, X_test12, y_train12, y_test12 = train_test_split(X, y, test_size=test_perc,

def f_syn_polarity12(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_12, ob
    score = np.array(cross_val_score(reg, X=X_train12, y=y_train12).mean())
    return operator * score

approx_12 = GPGO(surrogate_approx_12, Acquisition_new(util_approx), f_syn_polarity12, p
approx_12.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_approx_12 = approx_12.getResult()[0]
params_approx_12['max_depth'] = int(params_approx_12['max_depth'])
params_approx_12['min_child_weight'] = int(params_approx_12['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_approx_train12 = xgb.DMatrix(X_train12, y_train12)
```

```
dX_approx_train12 = xgb.DMatrix(X_train12, y_train12)
dX_approx_test12 = xgb.DMatrix(X_test12, y_test12)
model_approx_12 = xgb.train(params_approx_12, dX_approx_train12)
pred_approx_12 = model_approx_12.predict(dX_approx_test12)

rmse_approx_12 = np.sqrt(mean_squared_error(pred_approx_12, y_test12))
rmse_approx_12
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [1.54162842 7.40049697 6.         0.54321714 4.         0.11311747].     -0
init     [ 9.18747008  9.00714854 14.          0.97847467 11.          0.35544552].
init     [ 6.06083184  9.44225136 14.          0.95626942  5.          0.56910342].
init     [ 5.52037633  4.85377414  7.          0.97886436 17.          0.78810441].
init     [ 0.20809798  1.35210178  5.          0.65494879 16.          0.36062811].
1        [9.46555822 8.57190559 5.         0.50164398 5.         0.71992807].     -0
2        [9.04256367 2.61736915 8.         0.66026854 8.         0.14510453].     -0
3        [6.03751892 2.08855857 8.         0.88966175 1.         0.6215545 ].     -0
4        [ 0.24796255  2.18203944 14.          0.56497025 17.          0.63132662].
5        [ 1.93384153  7.13950146  8.          0.85480597 18.          0.33734734].
6        [ 0.40359854  2.22527636 10.          0.60258213 10.          0.38255809].
7        [ 0.28427394  4.67296732 14.          0.84909523  4.          0.99176009].
8        [ 7.63658847  0.39719075 14.          0.96199388 14.          0.84093877].
9        [ 0.50213582  8.87075    12.          0.71856843 12.          0.66120412].
10       [ 4.27921374  9.2199845   6.          0.67076861 12.          0.56605459].
11       [ 7.63578483  4.07501457 14.          0.97723751  1.          0.2033266 ].
12       [ 9.06259994  1.4172751   5.          0.81347212 19.          0.38881712].
13       [ 9.38461996  5.62749581 12.          0.98689844 17.          0.89710846].
14       [ 4.55964005  0.30434123  5.          0.52399968 11.          0.14256504].
15       [ 6.59943135  1.71178305 14.          0.69283152  8.          0.30459736].
16       [5.99294447 7.0114806  5.         0.86352024 2.         0.47030879].     -0
17       [ 9.63630565  6.83547158  8.          0.6195521  13.          0.11712353].
18       [0.93465239 0.11560545 5.         0.52904926 2.         0.78190279].     -0
19       [ 4.02953989  6.35914994 11.          0.73063178  8.          0.53793685].
20       [ 0.49442426  9.20422434 11.          0.70522852  1.          0.50168224].
4.548920198104285
```

```
### Bayesian optimization runs (x20): 'approx' Acquisition Function run number = 13

np.random.seed(run_num_13)
surrogate_approx_13 = dGaussianProcess(cov_func, optimize=opt)

X_train13, X_test13, y_train13, y_test13 = train_test_split(X, y, test_size=test_perc,

def f_syn_polarity13(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_13, ob
    score = np.array(cross_val_score(reg, X=X_train13, y=y_train13).mean())
    return operator * score

approx_13 = GPGO(surrogate_approx_13, Acquisition_new(util_approx), f_syn_polarity13, p
approx_13.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_approx_13 = approx_13.getResult()[0]
params_approx_13['max_depth'] = int(params_approx_13['max_depth'])
params_approx_13['min_child_weight'] = int(params_approx_13['min_child_weight'])

```

```
### Re-train with optimal parameters, run predictons:
dX_approx_train13 = xgb.DMatrix(X_train13, y_train13)
dX_approx_test13 = xgb.DMatrix(X_test13, y_test13)
model_approx_13 = xgb.train(params_approx_13, dX_approx_train13)
pred_approx_13 = model_approx_13.predict(dX_approx_test13)

rmse_approx_13 = np.sqrt(mean_squared_error(pred_approx_13, y_test13))
rmse_approx_13
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 7.77702411  2.3754122  11.         0.94649135 13.         0.7827256 ].
init     [ 7.51661514  6.07343344 11.         0.69402149 11.         0.13153287].
init     [ 2.98449471  0.58512492 10.         0.73579614 12.         0.33065195].
init     [ 3.47581215  0.0941277  11.         0.86143432  8.         0.58454932].
init     [ 4.70137857  6.24432527 10.         0.8149145  18.         0.10784416].
1        [1.1119361  5.43221306 6.         0.56899303 8.         0.32100319].     -0
2        [5.39023698 3.80105709 8.         0.54170057 1.         0.56798884].     -0
3        [ 0.5185863   5.23876151 13.         0.63798348  5.         0.7914799 ].
4        [ 9.65518672  0.13040633  6.         0.63296628 18.         0.44133279].
5        [ 9.80722669  7.22571611  7.         0.5026908  19.         0.92519376].
6        [ 6.75965929  9.42320667 14.         0.75932127  4.         0.51195623].
7        [9.95671825 0.88335607 5.         0.76593799 7.         0.60049246].     -0
8        [ 1.88898055  9.92199995 14.         0.53783103 12.         0.63359705].
9        [0.32121091 9.03384774 6.         0.79493663 1.         0.29330571].     -0
10       [ 9.64211232  3.05396831 11.         0.80784352  5.         0.67910498].
11       [ 1.60018805  0.54211528  7.         0.80910607 18.         0.55232873].
12       [ 6.1829314   9.53799326  5.         0.66916589 13.         0.57013155].
13       [7.99022981 8.23715587 5.         0.79172469 5.         0.39852784].     -0
14       [ 8.15066897  1.98276445 13.         0.64083395 19.         0.16153982].
15       [ 2.48128047  9.70146472  6.         0.59664322 19.         0.9781653 ].
16       [ 9.08793394  9.83715934 14.         0.60607659 14.         0.5136415 ].
17       [ 1.80857777  4.26197    14.         0.84061579 16.         0.40224246].
18       [ 5.1333583   3.10658752 14.         0.74854361  1.         0.92539511].
19       [ 0.65009462  9.91088996 13.         0.92448902  2.         0.37838494].
20       [ 0.18073363  6.71350851  7.         0.80658597 14.         0.6264676 ].
4.601162573826364
```

```
### Bayesian optimization runs (x20): 'approx' Acquisition Function run number = 14

np.random.seed(run_num_14)
surrogate_approx_14 = dGaussianProcess(cov_func, optimize=opt)

X_train14, X_test14, y_train14, y_test14 = train_test_split(X, y, test_size=test_perc,

def f_syn_polarity14(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_14, ob
    score = np.array(cross_val_score(reg, X=X_train14, y=y_train14).mean())
    return operator * score

approx_14 = GPGO(surrogate_approx_14, Acquisition_new(util_approx), f_syn_polarity14, p
approx_14.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_approx_14 = approx_14.getResult()[0]
params_approx_14['max_depth'] = int(params_approx_14['max_depth'])
params_approx_14['min_child_weight'] = int(params_approx_14['min_child_weight'])
```

```
params_approx_14['min_child_weight'] = int(params_approx_14['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_approx_train14 = xgb.DMatrix(X_train14, y_train14)
dX_approx_test14 = xgb.DMatrix(X_test14, y_test14)
model_approx_14 = xgb.train(params_approx_14, dX_approx_train14)
pred_approx_14 = model_approx_14.predict(dX_approx_test14)

rmse_approx_14 = np.sqrt(mean_squared_error(pred_approx_14, y_test14))
rmse_approx_14
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 5.13943344  7.73165052 12.          0.6831412  11.          0.37876233].
init     [ 9.57603739  5.13116712 14.          0.76959997 12.          0.71328228].
init     [5.34950319 2.47493539 5.          0.50293689 6.          0.29706373].     -0
init     [ 2.94506579  3.45329697  8.          0.87620946 14.          0.9783044 ].
init     [ 1.11811929  1.73004086  5.          0.73745288 12.          0.20586008].
1        [ 6.50637223  2.67617722 14.          0.53562507  1.          0.16862152].
2        [ 9.97732733  0.9008687  13.          0.65397817 19.          0.96533011].
3        [ 0.28409124  4.13353348 13.          0.96339983  6.          0.90100709].
4        [9.32373648 9.05676215 9.          0.53064322 3.          0.70657534].     -0
5        [ 9.52454394  8.82757271  9.          0.90064956 19.          0.16022914].
6        [ 3.61508571  7.70718733 10.          0.62042707  5.          0.12186292].
7        [ 0.9687803   2.15143442 14.          0.51651811 18.          0.56051657].
8        [ 9.40430013  0.15700131  7.          0.99940272 12.          0.43771306].
9        [ 8.97462379  3.07976653  6.          0.89115666 18.          0.90399549].
10       [ 2.08494663  9.70581169 14.          0.64307382  3.          0.10613693].
11       [ 1.55781918  0.46142569 10.          0.80240433 16.          0.22604037].
12       [ 8.32471637  8.64868248  5.          0.80719895 12.          0.64183859].
13       [ 1.40044045  8.51657075 10.          0.9376679  19.          0.10378975].
14       [0.73443351 4.9474333  6.          0.76758914 1.          0.38158758].     -0
15       [ 0.4418527   4.49961454  5.          0.81022894 19.          0.68885116].
16       [9.87159789 0.82301761 9.          0.69884198 1.          0.99856434].     -0
17       [ 9.4187571   0.09158132 12.          0.99701098  7.          0.20997236].
18       [ 2.97507046  9.85127832  5.          0.75729568 12.          0.23402253].
19       [ 6.6896573   9.49786322 14.          0.98343455 18.          0.89331088].
20       [ 7.50974882  6.12589601 14.          0.77913808  6.          0.27444251].
4.562117790499774
```

```
### Bayesian optimization runs (x20): 'approx' Acquisition Function run number = 15

np.random.seed(run_num_15)
surrogate_approx_15 = dGaussianProcess(cov_func, optimize=opt)

X_train15, X_test15, y_train15, y_test15 = train_test_split(X, y, test_size=test_perc,

def f_syn_polarity15(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_15, ob
    score = np.array(cross_val_score(reg, X=X_train15, y=y_train15).mean())
    return operator * score

approx_15 = GPGO(surrogate_approx_15, Acquisition_new(util_approx), f_syn_polarity15, p
approx_15.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_approx_15 = approx_15.getResult()[0]
```

```
params_approx_15['max_depth'] = int(params_approx_15['max_depth'])
params_approx_15['min_child_weight'] = int(params_approx_15['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_approx_train15 = xgb.DMatrix(X_train15, y_train15)
dX_approx_test15 = xgb.DMatrix(X_test15, y_test15)
model_approx_15 = xgb.train(params_approx_15, dX_approx_train15)
pred_approx_15 = model_approx_15.predict(dX_approx_test15)

rmse_approx_15 = np.sqrt(mean_squared_error(pred_approx_15, y_test15))
rmse_approx_15
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 8.48817697  1.78895925 12.         0.55549316  8.         0.93397854].
init     [ 0.24953032  8.22298097 12.         0.62494951 11.         0.12924598].
init     [ 5.02017228  5.50882771 11.         0.85295832 19.         0.13548008].
init     [2.0023081  9.98543403 7.        0.6295772  2.        0.526127  ].     -0
init     [ 5.09715306  9.45038417 11.         0.7388277  16.         0.22739973].
1        [ 0.29158961  4.9949242  12.         0.89124583  3.         0.67554049].
2        [3.68214008 4.55748717 6.        0.60488381 8.        0.88973248].     -0
3        [9.75991344 6.15203198 6.        0.65490407 1.        0.73816291].     -0
4        [ 9.51793103  9.55070381  6.         0.93315157 11.         0.40416985].
5        [ 6.65116837  8.16324548 14.         0.95750787  1.         0.74925927].
6        [ 0.41861043  0.05076637  5.         0.78940316 12.         0.77619725].
7        [ 0.44758083  1.21361479 14.         0.72053858 16.         0.73785377].
8        [ 9.36080884  1.0313168   5.         0.6330142  14.         0.51274968].
9        [ 3.89599673  7.59173972  5.         0.94509004 14.         0.68417344].
10       [4.3552321  3.34830177 9.        0.76229473 1.        0.13073039].     -0
11       [ 0.22313453  7.46320923  9.         0.98956429 19.         0.22753615].
12       [ 3.91025291  0.94746935  6.         0.83827135 18.         0.68489482].
13       [ 1.41127451  0.02455301 14.         0.99438541  9.         0.56009967].
14       [ 7.09557213  0.02461437 14.         0.76122797 15.         0.64719348].
15       [ 7.70971055  5.9598343  10.         0.83735899 18.         0.36826156].
16       [0.         0.         5.        0.67913459 4.52932381 0.6656685 ].     -0
17       [9.52146177 3.75389952 5.        0.5049997  7.        0.95818795].     -0
18       [ 8.77636678  8.74391722 14.         0.79691628 11.         0.9639423 ].
19       [ 2.27839391  3.48085254  9.         0.82265329 13.         0.31218776].
20       [10.   0.  15.   0.5  1.   1. ].         -0.47516221601210623   -0.47422405
4.589957692331628
```

```
### Bayesian optimization runs (x20): 'approx' Acquisition Function run number = 16

np.random.seed(run_num_16)
surrogate_approx_16 = dGaussianProcess(cov_func, optimize=opt)

X_train16, X_test16, y_train16, y_test16 = train_test_split(X, y, test_size=test_perc,

def f_syn_polarity16(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_16, ob
    score = np.array(cross_val_score(reg, X=X_train16, y=y_train16).mean())
    return operator * score

approx_16 = GPGO(surrogate_approx_16, Acquisition_new(util_approx), f_syn_polarity16, p
approx_16.run(max_iter = max_iter, init_evals = n_init) # run

```

```
### Return optimal parameters' set:
params_approx_16 = approx_16.getResult()[0]
params_approx_16['max_depth'] = int(params_approx_16['max_depth'])
params_approx_16['min_child_weight'] = int(params_approx_16['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_approx_train16 = xgb.DMatrix(X_train16, y_train16)
dX_approx_test16 = xgb.DMatrix(X_test16, y_test16)
model_approx_16 = xgb.train(params_approx_16, dX_approx_train16)
pred_approx_16 = model_approx_16.predict(dX_approx_test16)

rmse_approx_16 = np.sqrt(mean_squared_error(pred_approx_16, y_test16))
rmse_approx_16
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [2.23291079 5.23163341 6.         0.65430839 5.         0.30077285].     -0
init     [6.88726162 1.63731425 7.         0.97050543 2.         0.25392012].     -0
init     [ 5.94328983  5.6393473   5.          0.67602695 19.          0.42538144].
init     [ 0.88741148  3.08148142 14.          0.56043938  9.          0.27515386].
init     [ 2.74631586  1.30996118 11.          0.52160786  8.          0.27956463].
1        [ 7.8937256   1.5972923  14.          0.61610774 17.          0.78739284].
2        [ 9.01655783  8.21383177  9.          0.60772965 10.          0.9401803 ].
3        [ 4.35132073  9.89698316 12.          0.94137984 16.          0.57741056].
4        [ 3.38377852  9.31285251 11.          0.88244942  1.          0.67627774].
5        [ 0.02157337  9.97534925  5.          0.75404051 13.          0.40760752].
6        [ 0.19317903  0.6596816   6.          0.86233301 15.          0.41906313].
7        [ 9.2502617   3.00821528  6.          0.66523104 13.          0.86141057].
8        [ 1.22130867  0.64008351 12.          0.59515137 19.          0.65193522].
9        [ 9.94620544  1.66561851 11.          0.64793653  8.          0.53536185].
10       [8.12731949 9.23712019 5.         0.71442381 2.         0.11528188].     -0
11       [ 8.63952355  8.87914214 14.          0.57943185  3.          0.83429859].
12       [ 2.23523952  3.62733473 12.          0.51556206  2.          0.2036675 ].
13       [ 7.72830432  0.56848968 14.          0.94940741  3.          0.61543829].
14       [2.43756281e-03 1.86203502e-01 1.10000000e+01 5.84519618e-01
 1.30000000e+01 2.30149841e-01].          -0.7072748319481598    -0.4830822737254163
15       [ 3.22380477  8.8210053  12.          0.91827732  7.          0.1365336 ].
16       [ 9.28336661  6.57187836 11.          0.54297007 17.          0.56210295].
17       [ 0.03255143  5.62206767 10.          0.695628   16.          0.90161277].
18       [ 0.06909596  9.96923496  8.          0.86088515 19.          0.34694652].
19       [ 6.18013749  9.69222015  6.          0.87704216 14.          0.79491401].
20       [ 6.23198582  4.61080248 11.          0.74322916 13.          0.31928721].
4.9062523492935854
```

```
### Bayesian optimization runs (x20): 'approx' Acquisition Function run number = 17

np.random.seed(run_num_17)
surrogate_approx_17 = dGaussianProcess(cov_func, optimize=opt)

X_train17, X_test17, y_train17, y_test17 = train_test_split(X, y, test_size=test_perc,

def f_syn_polarity17(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_17, ob
    score = np.array(cross_val_score(reg, X=X_train17, y=y_train17).mean())
    return operator * score

approx_17 = GPGO(surrogate_approx_17, Acquisition_new(util_approx), f_syn_polarity17, p
```

```
approx_17 = dGP(surrogate_approx_17, Acquisition_new(util_approx), f_syn_polarity17, p
approx_17.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_approx_17 = approx_17.getResult()[0]
params_approx_17['max_depth'] = int(params_approx_17['max_depth'])
params_approx_17['min_child_weight'] = int(params_approx_17['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_approx_train17 = xgb.DMatrix(X_train17, y_train17)
dX_approx_test17 = xgb.DMatrix(X_test17, y_test17)
model_approx_17 = xgb.train(params_approx_17, dX_approx_train17)
pred_approx_17 = model_approx_17.predict(dX_approx_test17)

rmse_approx_17 = np.sqrt(mean_squared_error(pred_approx_17, y_test17))
rmse_approx_17
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 2.94665003  5.30586756 11.          0.94443241 14.          0.80828691].
init     [ 6.56333522  6.37520896 12.          0.81487881 18.          0.42203224].
init     [ 9.45683187  0.6004468  11.          0.5171566  10.          0.53881211].
init     [2.72705857 1.19063434 6.          0.74176431 6.          0.10101151].     -0
init     [ 4.77631812  5.24671297 13.          0.66254476 19.          0.36708086].
1        [ 0.65702322  5.79284078 13.          0.75136902  1.          0.30306068].
2        [ 6.93446178  8.68032298 13.          0.78195789  7.          0.91906958].
3        [9.72843652 3.88893279 9.          0.6901555  1.          0.31608219].     -0
4        [ 9.65057736  8.52725784  5.          0.68420234 13.          0.40008732].
5        [ 4.97204887  2.40072226  5.          0.54268748 19.          0.30995407].
6        [0.12174033 8.73496008 5.          0.89827646 5.          0.85354798].     -0
7        [ 2.91443079  0.16723755 13.          0.598201    6.          0.91729605].
8        [7.20615247 9.36901627 6.          0.85465034 7.          0.6878262 ].     -0
9        [ 9.02586164  0.59354638 10.          0.86038693 18.          0.90794111].
10       [ 0.          5.95382041  5.          0.5        11.37472151  0.1       ].
11       [ 0.12410542  7.60180472 13.          0.97425003  8.          0.76245421].
12       [ 2.44282715  9.71851747  7.          0.86792183 17.          0.93015556].
13       [9.59190042 1.26233772 5.          0.55398722 6.          0.46259714].     -0
14       [ 9.06002681  9.03779351 14.          0.60614154  1.          0.16421461].
15       [ 0.59590325  0.30071901  8.          0.9148403  16.          0.62633871].
16       [ 8.69529605  0.27226865 14.          0.90636352  1.          0.39638219].
17       [ 5.5997101   2.80089542 11.          0.91397635  4.          0.58148876].
18       [ 0.          6.57711107 10.28785265  0.5        20.          0.1       ].
19       [ 9.24004771  6.36004976 11.          0.81644504 12.          0.66235401].
20       [10.         10.          5.00003108  1.          1.          1.        ].
4.758964442352727
```

```
### Bayesian optimization runs (x20): 'approx' Acquisition Function run number = 18

np.random.seed(run_num_18)
surrogate_approx_18 = dGaussianProcess(cov_func, optimize=opt)

X_train18, X_test18, y_train18, y_test18 = train_test_split(X, y, test_size=test_perc,

def f_syn_polarity18(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_11, ob
    score = np.array(cross_val_score(reg, X=X_train18, y=y_train18).mean())
    return operator * score
```

```
13
14 approx_18 = GPGO(surrogate_approx_18, Acquisition_new(util_approx), f_syn_polarity18, p
15 approx_18.run(max_iter = max_iter, init_evals = n_init) # run
16
17 ### Return optimal parameters' set:
18 params_approx_18 = approx_18.getResult()[0]
19 params_approx_18['max_depth'] = int(params_approx_18['max_depth'])
20 params_approx_18['min_child_weight'] = int(params_approx_18['min_child_weight'])
21
22 ### Re-train with optimal parameters, run predictons:
23 dX_approx_train18 = xgb.DMatrix(X_train18, y_train18)
24 dX_approx_test18 = xgb.DMatrix(X_test18, y_test18)
25 model_approx_18 = xgb.train(params_approx_18, dX_approx_train18)
26 pred_approx_18 = model_approx_18.predict(dX_approx_test18)
27
28 rmse_approx_18 = np.sqrt(mean_squared_error(pred_approx_18, y_test18))
29 rmse_approx_18
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [6.50374242 5.05453374 6.         0.59092011 3.         0.28357516].     -0
init     [0.11506734 4.26891483 9.         0.81785956 5.         0.63489043].     -0
init     [ 2.8861259   6.35547834 11.          0.64267955 14.          0.27877092].
init     [6.57189031 6.99655629 8.         0.63235896 4.         0.52894035].     -0
init     [ 6.66600348  2.11312037 14.          0.74363461  4.          0.73174558].
1        [ 8.67093232  0.11649132  5.          0.92962202 15.          0.53672863].
2        [ 8.43851229  2.41114508 13.          0.75771586 19.          0.86905071].
3        [ 9.44281001  9.01534322  7.          0.99142432 16.          0.37631199].
4        [ 3.19538294  9.91737336 14.          0.7976317   5.          0.25704487].
5        [ 1.97643014  8.37982471  5.          0.63246176 17.          0.45403539].
6        [ 1.26601315  0.31299408  6.          0.95296222 16.          0.13649883].
7        [ 1.18347798  2.03195078 14.          0.62549517 10.          0.6517083 ].
8        [6.72039962 1.0287777  9.         0.62848622 9.         0.70815446].     -0
9        [ 7.49192948  9.60283663 14.          0.93718151 19.          0.24625933].
10       [ 3.63870552  9.73349763  7.          0.73625258 11.          0.87968363].
11       [9.98653758 8.80568206 9.         0.86984433 9.         0.78984159].     -0
12       [ 1.68019344  0.20490035 13.          0.89332378 19.          0.17761947].
13       [0.58655573 8.89860432 5.         0.62593909 1.         0.46819727].     -0
14       [ 4.25762222  4.02301922  9.          0.6129246  19.          0.59981465].
15       [ 5.93963174  0.35038192 14.          0.99474968 14.          0.46057279].
16       [2.5556895  0.91787864 5.         0.57129972 6.         0.40466558].     -0
17       [ 8.60576508  8.57930928 14.          0.78656855  7.          0.59728424].
18       [ 9.56072091  9.1368036  14.          0.68050527  1.          0.47597207].
19       [ 0.          6.72821976 10.97306698  1.         19.00755008  0.1       ].
20       [ 2.08722767  3.98919754  5.          0.76742571 11.          0.93277603].
4.746067780922308
```

```
1 ### Bayesian optimization runs (x20): 'approx' Acquisition Function run number = 19
2
3 np.random.seed(run_num_19)
4 surrogate_approx_19 = dGaussianProcess(cov_func, optimize=opt)
5
6 X_train19, X_test19, y_train19, y_test19 = train_test_split(X, y, test_size=test_perc,
7
8 def f_syn_polarity19(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
9     reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
10          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_19, ob
```

```
    score = np.array(cross_val_score(reg, X=X_train19, y=y_train19).mean())
    return operator * score

approx_19 = GPGO(surrogate_approx_19, Acquisition_new(util_approx), f_syn_polarity19, p
approx_19.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_approx_19 = approx_19.getResult()[0]
params_approx_19['max_depth'] = int(params_approx_19['max_depth'])
params_approx_19['min_child_weight'] = int(params_approx_19['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_approx_train19 = xgb.DMatrix(X_train19, y_train19)
dX_approx_test19 = xgb.DMatrix(X_test19, y_test19)
model_approx_19 = xgb.train(params_approx_19, dX_approx_train19)
pred_approx_19 = model_approx_19.predict(dX_approx_test19)

rmse_approx_19 = np.sqrt(mean_squared_error(pred_approx_19, y_test19))
rmse_approx_19
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 0.97533602  7.61249717 13.          0.85765469 11.          0.39830191].
init     [ 0.82999565  6.71977081  6.          0.50407413 19.          0.67209466].
init     [ 2.15923256  5.49027432 12.          0.52588686 10.          0.20235326].
init     [4.99659267 1.52108422 6.         0.73481085 4.         0.71949465].     -0
init     [ 3.72927156  9.46160045  5.          0.80554614 18.          0.97708466].
1        [ 8.33060043  1.42030563  8.          0.92863724 14.          0.78606141].
2        [ 4.70068371  4.9755295  14.          0.98901029  1.          0.96039614].
3        [9.07948237 9.55063617 7.         0.80962147 4.         0.34142584].     -0
4        [ 3.65000245  2.90359952 13.          0.98940034 19.          0.29019455].
5        [0.25768796 8.33414072 6.         0.99948369 4.         0.66680619].     -0
6        [ 8.89243569  4.2136102  10.          0.88870164  8.          0.77683659].
7        [ 9.20734788  9.40702602 11.          0.68264617 14.          0.68723167].
8        [1.60895472e-01 8.27074864e-03 1.30000000e+01 6.91893018e-01
 5.00000000e+00 4.60327119e-01].          -0.5676975299111877    -0.454763872590455
9        [ 0.32873959  7.7155114   5.          0.56871173 11.          0.34471029].
10       [ 6.55489773  0.06438745 14.          0.53351105 11.          0.76391016].
11       [ 9.32390837  9.51845123 14.          0.71072327  2.          0.55726506].
12       [ 4.31405249  0.92741236  7.          0.67333637 19.          0.97097788].
13       [ 4.44431879  9.49168646 13.          0.64581011 19.          0.4702202 ].
14       [ 5.91102876  9.59864462 12.          0.78362129  7.          0.97877384].
15       [0.22611985 0.1170465  7.         0.55795985 9.         0.86758401].     -0
16       [ 9.0780433   8.49155787  5.          0.8925855  13.          0.38273904].
17       [ 9.89996921  0.59570014 13.          0.67906413  1.          0.37793763].
18       [ 0.65734769  9.79668398 13.          0.57710479  1.          0.50646708].
19       [ 0.17032432  0.5511349   8.          0.8772351  15.          0.24483614].
20       [ 3.82670531  7.06245628  8.          0.7747773  14.          0.11384763].
4.63424117920896
```

```
### Bayesian optimization runs (x20): 'approx' Acquisition Function run number = 20

np.random.seed(run_num_20)
surrogate_approx_20 = dGaussianProcess(cov_func, optimize=opt)

X_train20, X_test20, y_train20, y_test20 = train_test_split(X, y, test_size=test_perc,

def f_syn_polarity20(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
```

```
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_20, ob
    score = np.array(cross_val_score(reg, X=X_train20, y=y_train20).mean())
    return operator * score

approx_20 = GPGO(surrogate_approx_20, Acquisition_new(util_approx), f_syn_polarity20, p
approx_20.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_approx_20 = approx_20.getResult()[0]
params_approx_20['max_depth'] = int(params_approx_20['max_depth'])
params_approx_20['min_child_weight'] = int(params_approx_20['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_approx_train20 = xgb.DMatrix(X_train20, y_train20)
dX_approx_test20 = xgb.DMatrix(X_test20, y_test20)
model_approx_20 = xgb.train(params_approx_20, dX_approx_train20)
pred_approx_20 = model_approx_20.predict(dX_approx_test20)

rmse_approx_20 = np.sqrt(mean_squared_error(pred_approx_20, y_test20))
rmse_approx_20
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 5.88130801  8.97713728 14.         0.81074445  8.         0.95540649].
init     [6.72865655 0.41173329 8.         0.6361582  7.         0.76174061].     -0
init     [ 4.77387703  8.66202323 10.         0.51833215  7.         0.10123387].
init     [ 5.75489985  4.74524381  8.         0.78084343 15.         0.26643049].
init     [ 4.53444     4.47342833  8.         0.91974896 18.         0.35997552].
1        [ 7.96566073  7.15509535  7.         0.79906691 11.         0.34132075].
2        [0.0691652  5.34850007 7.         0.63667652 1.         0.21496993].     -0
3        [ 3.00704909  2.42524876 14.         0.95062509 15.         0.83595087].
4        [ 7.32450119  0.97088138 14.         0.62864562  2.         0.93189305].
5        [8.0846212  5.99993376 6.         0.83941375 1.         0.46362124].     -0
6        [ 4.85539645  9.01721781 14.         0.60995232 19.         0.22249343].
7        [ 0.72788527  2.26655356 10.         0.97273032 15.         0.85843758].
8        [ 7.29847873  0.61439441  5.         0.60353619 15.         0.628184  ].
9        [ 0.05422088  9.41967109  9.         0.63564274 14.         0.96864948].
10       [ 9.46105078  8.51558179 14.         0.6234267   1.         0.59075489].
11       [ 3.28444135  4.88068826 13.         0.83217983  5.         0.1064095 ].
12       [ 9.95790482  0.39582549 14.         0.73682131 13.         0.88500223].
13       [ 8.98143836  8.7693897  12.         0.89468403 12.         0.66552576].
14       [ 9.53401947  2.04639571 13.         0.59498138 19.         0.43345618].
15       [ 5.37444991  0.3056877   9.         0.99777328 15.         0.5903115 ].
16       [1.01173572 1.57355816 5.         0.66922967 6.         0.97562515].     -0
17       [ 9.44977056  8.08706416 10.         0.57562031 19.         0.10040302].
18       [ 0.95166608  0.72165502 12.         0.68549843  9.         0.11447959].
19       [ 0.72440064  9.37673317 12.         0.5979442   2.         0.11910674].
20       [ 1.39513001  1.37149056  5.         0.56200696 12.         0.29219492].
4.454763173494824
```

```
end_approx = time.time()
end_approx

time_approx = end_approx - start_approx
time_approx

```

```
start_exact = time.time()
start_exact
```

```
1623252576.0246754
```

```
### Bayesian optimization runs (x20): 'exact' Acquisition Function run number = 1

np.random.seed(run_num_1)
surrogate_exact_1 = dGaussianProcess(cov_func, optimize=opt)

X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=test_perc, rand

def f_syn_polarity1(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_1, obj
    score = np.array(cross_val_score(reg, X=X_train1, y=y_train1).mean())
    return operator * score

exact_1 = dGPGO(surrogate_exact_1, Acquisition_new(util_exact), f_syn_polarity1, param,
exact_1.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_exact_1 = exact_1.getResult()[0]
params_exact_1['max_depth'] = int(params_exact_1['max_depth'])
params_exact_1['min_child_weight'] = int(params_exact_1['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_exact_train1 = xgb.DMatrix(X_train1, y_train1)
dX_exact_test1 = xgb.DMatrix(X_test1, y_test1)
model_exact_1 = xgb.train(params_exact_1, dX_exact_train1)
pred_exact_1 = model_exact_1.predict(dX_exact_test1)

rmse_exact_1 = np.sqrt(mean_squared_error(pred_exact_1, y_test1))
rmse_exact_1
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [5.48813504 7.15189366 8.         0.92897281 8.         0.48128932].     -0
init     [ 6.45894113  4.37587211 11.          0.52835649 13.          0.44509737].
init     [ 7.91725038  5.2889492  13.          0.6963924  14.          0.40365654].
init     [ 6.48171872  3.6824154  10.          0.88907838 16.          0.88307853].
init     [4.73608045 8.00910752 8.         0.83943977 8.         0.67592892].     -0
1        [ 0.96098408  9.76459465  7.          0.75481219 17.          0.64436097].
2        [ 5.13759733  2.22657933 12.          0.58106013  2.          0.92007745].
3        [9.58067178 9.65734278 7.         0.88193436 1.         0.38223155].     -0
4        [0.  0.  5.  0.5 1.  0.1].      -0.6779967301528039    -0.4614357236027627
5        [ 0.12917683  7.27788751 10.          0.94371091  2.          0.12091296].
6        [ 8.87166351  9.3367646   5.          0.58200219 19.          0.4055524 ].
7        [ 0.51228404  8.90605177 14.          0.7949699  11.          0.73913262].
8        [ 2.05150398  0.53599727  5.          0.71551734 11.          0.37377971].
9        [8.38797278 0.6003286  6.         0.6181346  1.         0.60370114].     -0
10       [ 9.06530919  9.40796401 14.          0.73452132  4.          0.22638202].
11       [ 2.92761431  0.02841708 13.          0.97950295  9.          0.31091424].
12       [9.14092793 0.64739809 8.         0.90967035 8.         0.70435457].     -0
13       [ 1.99180311  1.76156949  6.          0.52474573 18.          0.69397695].
14       [ 1.47165443  0.32474578 13.          0.93757169 15.          0.40230457].
15       [ 3.33998723  9.61997442 13.          0.87678219 17.          0.47112812].
16       [ 0.59268574  5.88934857  8.          0.90451848 12.          0.19426621].
```

```
17       [ 7.96942817  9.76181769  7.          0.91046857 13.          0.32455735].
18       [4.22240748 6.41687796 5.         0.84764325 1.         0.63972487].     -0
19       [ 8.8155061   8.22817745 14.          0.53858872 19.          0.2617363 ].
20       [ 0.37157734  3.11392041 14.          0.79304737  5.          0.73468784].
4.667222167746296
```

```
### Bayesian optimization runs (x20): 'exact' Acquisition Function run number = 2

np.random.seed(run_num_2)
surrogate_exact_2 = dGaussianProcess(cov_func, optimize=opt)

X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, test_size=test_perc, rand

def f_syn_polarity2(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_2, obj
    score = np.array(cross_val_score(reg, X=X_train2, y=y_train2).mean())
    return operator * score

exact_2 = dGPGO(surrogate_exact_2, Acquisition_new(util_exact), f_syn_polarity2, param,
exact_2.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_exact_2 = exact_2.getResult()[0]
params_exact_2['max_depth'] = int(params_exact_2['max_depth'])
params_exact_2['min_child_weight'] = int(params_exact_2['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_exact_train2 = xgb.DMatrix(X_train2, y_train2)
dX_exact_test2 = xgb.DMatrix(X_test2, y_test2)
model_exact_2 = xgb.train(params_exact_2, dX_exact_train2)
pred_exact_2 = model_exact_2.predict(dX_exact_test2)

rmse_exact_2 = np.sqrt(mean_squared_error(pred_exact_2, y_test2))
rmse_exact_2
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 4.35994902  0.25926232 11.          0.97386531 12.          0.47833102].
init     [ 3.30334821  2.04648634 10.          0.55997527  6.          0.71472339].
init     [ 4.9856117   5.86796978  8.          0.89266757 11.          0.59158659].
init     [ 4.07307832  1.76984624 13.          0.75262305  7.          0.35908193].
init     [ 1.16193318  1.81727038  9.          0.79837265 19.          0.29965165].
1        [ 9.41115874  8.16019152 11.          0.70945365  2.          0.28765225].
2        [ 8.78180153  6.61060882 12.          0.91523653 18.          0.29687212].
3        [ 0.66591974  9.26661294 14.          0.96342421 18.          0.94909068].
4        [0.53023554 8.79041977 6.         0.85342606 1.         0.93968064].     -0
5        [6.47584802 1.65960359 8.         0.80737784 1.         0.90602123].     -0
6        [ 2.29390808  9.72620131 13.          0.87738596  7.          0.2041919 ].
7        [ 9.77744834  2.26597384  5.          0.98618685 19.          0.58642903].
8        [7.36877801 8.87815651 5.         0.56972286 4.         0.80625064].     -0
9        [ 5.76886466  6.30636441 11.          0.85075313  6.          0.94564556].
10       [0.  0.  5.  0.5 1.  0.1].       -0.6760687561093285    -0.3879430352217851
11       [ 8.66397635  9.94226364  5.          0.61754568 19.          0.46861962].
12       [ 2.21750241  8.1937508   6.          0.88756269 17.          0.73205426].
13       [ 0.          0.          5.          0.5        10.23889203  0.1       ].
14       [ 2.61078484  8.48438058 14.          0.79784401 12.          0.37480602].
```

```
15       [ 9.63046012  9.46551843 14.          0.83717129 10.          0.78268661].
16       [ 8.39253634  0.41975132  5.          0.53583299 10.          0.93462258].
17       [ 1.36729794  4.89791321 13.          0.50996816  1.          0.64172429].
18       [ 8.76615518  1.51167594 13.          0.97549904  3.          0.62972174].
19       [ 5.41002785  1.04159958  6.36505818  0.5        16.36505818  0.1       ].
20       [ 8.94175283  9.74417102 10.          0.87790285 14.          0.24925416].
4.600295852493361
```

```
### Bayesian optimization runs (x20): 'exact' Acquisition Function run number = 3

np.random.seed(run_num_3)
surrogate_exact_3 = dGaussianProcess(cov_func, optimize=opt)

X_train3, X_test3, y_train3, y_test3 = train_test_split(X, y, test_size=test_perc, rand

def f_syn_polarity3(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_3, obj
    score = np.array(cross_val_score(reg, X=X_train3, y=y_train3).mean())
    return operator * score

exact_3 = dGPGO(surrogate_exact_3, Acquisition_new(util_exact), f_syn_polarity3, param,
exact_3.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_exact_3 = exact_3.getResult()[0]
params_exact_3['max_depth'] = int(params_exact_3['max_depth'])
params_exact_3['min_child_weight'] = int(params_exact_3['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_exact_train3 = xgb.DMatrix(X_train3, y_train3)
dX_exact_test3 = xgb.DMatrix(X_test3, y_test3)
model_exact_3 = xgb.train(params_exact_3, dX_exact_train3)
pred_exact_3 = model_exact_3.predict(dX_exact_test3)

rmse_exact_3 = np.sqrt(mean_squared_error(pred_exact_3, y_test3))
rmse_exact_3
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 5.50797903  7.08147823 13.          0.56066429 11.          0.11687321].
init     [ 0.40630737  2.47888297 11.          0.72040492 13.          0.23083313].
init     [ 4.53172301  2.15577008 11.          0.74631796  2.          0.60296868].
init     [ 2.59252447  4.15101197 13.          0.79330998  8.          0.24118096].
init     [ 5.44649018  7.80314765 10.          0.62879264 18.          0.44917413].
1        [1.56262424 9.7795241  5.          0.91450054 5.          0.53102391].     -0
2        [ 8.93142368  1.52910591 13.          0.84039318 17.          0.60846833].
3        [ 6.38594331  1.19109066  5.          0.81189053 13.          0.59164768].
4        [ 1.02918863  9.32189805 13.          0.88333707  1.          0.86998588].
5        [ 9.74929058  1.51205926 11.          0.50025602  8.          0.46132437].
6        [ 9.45052852  8.62641484  7.          0.79615518 14.          0.32790361].
7        [ 8.92744991  9.09956287 12.          0.74944313  3.          0.11030352].
8        [6.90239429 4.38257693 5.          0.77543741 6.          0.209803  ].     -0
9        [ 3.41151761  3.44257275  8.          0.91039266 17.          0.41829904].
10       [9.43215663 9.14652183 5.          0.95117899 1.          0.516629  ].     -0
11       [ 3.07505056  9.78732799  6.          0.96847152 12.          0.34771766].
12       [0.20657469 2.03837081 5.          0.940951   6.          0.11008073].     -0
```

```
13       [ 0.63366318  7.00411349 14.          0.89817768 18.          0.82387154].
14       [ 9.9224789   2.12085607  6.          0.82807413 19.          0.27539821].
15       [ 5.7586577   1.42812945 10.          0.74022438  8.          0.79053147].
16       [ 7.41867878  4.36581543 14.          0.99315379  6.          0.89117614].
17       [3.69167267 6.10773586 6.         0.9974767  1.         0.55722723].     -0
18       [ 2.77081064  4.33888713  7.          0.86748866 10.          0.91605352].
19       [ 0.2848593   7.24262219  6.          0.70597553 19.          0.97089957].
20       [8.69445334 9.32412616 7.         0.88093126 7.         0.17908776].     -0
4.649527586702978
```

```
### Bayesian optimization runs (x20): 'exact' Acquisition Function run number = 4

np.random.seed(run_num_4)
surrogate_exact_4 = dGaussianProcess(cov_func, optimize=opt)

X_train4, X_test4, y_train4, y_test4 = train_test_split(X, y, test_size=test_perc, rand

def f_syn_polarity4(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_4, obj
    score = np.array(cross_val_score(reg, X=X_train4, y=y_train4).mean())
    return operator * score

exact_4 = dGPGO(surrogate_exact_4, Acquisition_new(util_exact), f_syn_polarity4, param,
exact_4.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_exact_4 = exact_4.getResult()[0]
params_exact_4['max_depth'] = int(params_exact_4['max_depth'])
params_exact_4['min_child_weight'] = int(params_exact_4['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_exact_train4 = xgb.DMatrix(X_train4, y_train4)
dX_exact_test4 = xgb.DMatrix(X_test4, y_test4)
model_exact_4 = xgb.train(params_exact_4, dX_exact_train4)
pred_exact_4 = model_exact_4.predict(dX_exact_test4)

rmse_exact_4 = np.sqrt(mean_squared_error(pred_exact_4, y_test4))
rmse_exact_4
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [9.67029839 5.47232249 6.         0.92781047 9.         0.72795594].     -0
init     [ 2.16089496  9.76274455 12.          0.62649118  9.          0.66966679].
init     [ 0.05159149  5.72356491  9.          0.99170034 10.          0.10808749].
init     [ 3.86571283  0.44160058 10.          0.90553105 18.          0.95407958].
init     [ 7.86305986  8.66289299  6.          0.53285477 14.          0.25117497].
1        [ 8.45443649  8.61014312 11.          0.83475494  1.          0.14018305].
2        [ 0.77431146  1.96668116 12.          0.50723361  3.          0.74768925].
3        [2.27858743 6.23199766 5.         0.58705984 2.         0.80794289].     -0
4        [ 6.832625    9.87635293 14.          0.84450885 19.          0.20389666].
5        [ 7.37255369  2.03491596 13.          0.8921741   9.          0.46934318].
6        [ 0.05992751  6.06320143 14.          0.89322475 17.          0.32211096].
7        [ 0.79250634  6.36332745  6.          0.84703891 17.          0.8628914 ].
8        [9.26767626 0.09691703 5.         0.59554562 3.         0.95249041].     -0
9        [ 9.93824172  2.34876498  7.          0.94210707 16.          0.6947317 ].
10       [3.43076773 0.51115291 6.         0.80398076 7.         0.163561  ].     -0
```

```
11       [8.12039932 9.82838311 5.         0.6851891  3.         0.87462757].     -0
12       [ 6.03647398  5.02077859  7.         0.51964174 12.         0.90396407].
13       [ 9.03174101  2.34471053 13.         0.52042845  3.         0.20136175].
14       [ 9.29091353  0.24848851 14.         0.65742606 17.         0.33836791].
15       [3.19488299 9.73527155 6.         0.65266597 9.         0.60657999].     -0
16       [ 2.02212851  9.9407933  10.         0.71009222  3.         0.59925198].
17       [0.         0.         5.24159865 0.5        1.         0.1        ].     -0
18       [ 6.65760625  2.35253217 11.         0.58483338 14.         0.21088543].
19       [ 0.9141999   0.89707594 12.         0.91410151 11.         0.69348379].
20       [ 6.39825002  5.88858918  9.         0.67432081 18.         0.813834  ].
5.04081354533672
```

```
### Bayesian optimization runs (x20): 'exact' Acquisition Function run number = 5

np.random.seed(run_num_5)
surrogate_exact_5 = dGaussianProcess(cov_func, optimize=opt)

X_train5, X_test5, y_train5, y_test5 = train_test_split(X, y, test_size=test_perc, rand

def f_syn_polarity5(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_5, obj
    score = np.array(cross_val_score(reg, X=X_train5, y=y_train5).mean())
    return operator * score

exact_5 = dGPGO(surrogate_exact_5, Acquisition_new(util_exact), f_syn_polarity5, param,
exact_5.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_exact_5 = exact_5.getResult()[0]
params_exact_5['max_depth'] = int(params_exact_5['max_depth'])
params_exact_5['min_child_weight'] = int(params_exact_5['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_exact_train5 = xgb.DMatrix(X_train5, y_train5)
dX_exact_test5 = xgb.DMatrix(X_test5, y_test5)
model_exact_5 = xgb.train(params_exact_5, dX_exact_train5)
pred_exact_5 = model_exact_5.predict(dX_exact_test5)

rmse_exact_5 = np.sqrt(mean_squared_error(pred_exact_5, y_test5))
rmse_exact_5
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 2.21993171  8.70732306 11.         0.68186845 10.         0.53957007].
init     [ 6.11743863  7.65907856  5.         0.64840025 16.         0.82745351].
init     [ 6.49458883  8.19472793  6.         0.93996852 19.         0.36647194].
init     [ 6.28787909  5.7983781   6.         0.63290956 17.         0.18402673].
init     [8.26554249 8.33492742 9.         0.97900675 3.         0.26957319].     -0
1        [1.95474956 1.21548467 5.         0.65548996 6.         0.3261206 ].     -0
2        [ 7.93606002  2.5573513  12.         0.51144437  9.         0.97705142].
3        [ 6.12310163  2.21013771 14.         0.74740665 18.         0.42989776].
4        [ 2.51924113  6.01871441 14.         0.73887463  3.         0.83338103].
5        [ 0.45511658  1.05531983  8.         0.5285983  14.         0.24399377].
6        [ 7.94120484  0.92506262 13.         0.63913817  1.         0.95746958].
7        [ 3.55127355  9.21511435 13.         0.7432394  19.         0.48251811].
8        [9.06798103 6.21762396 5.         0.8435399  9.         0.45913475].     -0
```

```
9       [ 7.68019847  0.14272251  5.          0.84271757 13.          0.46064437].
10      [1.24717977 9.40645683 5.         0.65383928 3.         0.68852908].     -0
11      [ 9.92907009  9.56290601 12.          0.96474935 11.          0.11977881].
12      [9.35577232 0.06717833 6.         0.63643955 3.         0.33760798].     -0
13      [ 0.26037909  7.96430976  5.          0.57300432 18.          0.65170572].
14      [ 1.42238814  0.69874859 12.          0.81050321  6.          0.18951707].
15      [2.03762865 4.34961943 8.         0.53797581 1.         0.36410352].     -0
16      [ 0.20979055  3.48699735 13.          0.91075437 12.          0.42671314].
17      [ 2.57490322  0.33895391 10.          0.88927583 19.          0.31386234].
18      [ 9.2875561   9.15624524 13.          0.91867239 19.          0.68652241].
19      [ 0.31616346  9.22865905  5.          0.69019314 12.          0.52233884].
20      [ 9.94984248  2.37676311  9.          0.96621815 19.          0.48539202].
4.8115063045669055
```

```
### Bayesian optimization runs (x20): 'exact' Acquisition Function run number = 6

np.random.seed(run_num_6)
surrogate_exact_6 = dGaussianProcess(cov_func, optimize=opt)

X_train6, X_test6, y_train6, y_test6 = train_test_split(X, y, test_size=test_perc, rand

def f_syn_polarity6(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_6, obj
    score = np.array(cross_val_score(reg, X=X_train6, y=y_train6).mean())
    return operator * score

exact_6 = dGPGO(surrogate_exact_6, Acquisition_new(util_exact), f_syn_polarity6, param,
exact_6.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_exact_6 = exact_6.getResult()[0]
params_exact_6['max_depth'] = int(params_exact_6['max_depth'])
params_exact_6['min_child_weight'] = int(params_exact_6['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_exact_train6 = xgb.DMatrix(X_train6, y_train6)
dX_exact_test6 = xgb.DMatrix(X_test6, y_test6)
model_exact_6 = xgb.train(params_exact_6, dX_exact_train6)
pred_exact_6 = model_exact_6.predict(dX_exact_test6)

rmse_exact_6 = np.sqrt(mean_squared_error(pred_exact_6, y_test6))
rmse_exact_6
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [8.92860151 3.31979805 5.         0.99251441 2.         0.57683563].     -0
init     [4.18807429 3.35407849 9.         0.87750649 3.         0.56623277].     -0
init     [ 5.788586    6.45355096 14.          0.70660047 12.          0.82154882].
init     [4.58184578 6.73834679 5.         0.90108528 3.         0.65482895].     -0
init     [ 4.42510505  5.75952352 14.          0.97882365 15.          0.29525604].
1        [0.         0.         5.         0.5        9.28951254 0.1       ].     -0
2        [ 9.24343066  2.50578958  7.          0.73023742 13.          0.6940436 ].
3        [8.18334854 9.97104849 8.         0.91869016 8.         0.37558961].     -0
4        [ 8.54451719  6.55901746 13.          0.7507391   5.          0.45426156].
5        [ 1.35461816  3.68867636  7.          0.97358458 19.          0.99760691].
6        [ 0.58299146  9.62458538  9.          0.94830232 11.          0.21492907].
```

```
7       [ 7.23238967  9.57077424  9.          0.98910406 19.          0.89873182].
8       [ 9.32420466  6.39616005 13.          0.93300527 17.          0.34904443].
9       [ 2.3330023   9.76264059 12.          0.88161689  4.          0.98638718].
10      [ 6.80447195  6.69417926  5.          0.74586765 13.          0.74060763].
11      [ 0.56219975  9.42928268 11.          0.51903033 19.          0.52214278].
12      [ 2.15833867  2.09607443 11.          0.61964692  9.          0.5751364 ].
13      [ 5.43877488  0.17504551 13.          0.58721411 19.          0.60115325].
14      [0.  0.  5.  0.5 1.  0.1].       -0.6973217959617746    -0.4572803868503328
15      [ 7.781102    0.66670976 14.          0.78656914  6.          0.19023326].
16      [ 9.16380122  1.57669417  6.          0.94496183 19.          0.68061955].
17      [ 2.24051343  5.25045532  5.          0.81492378 10.          0.3308004 ].
18      [ 7.23381367  0.21465732 13.          0.75484136 13.          0.44717401].
19      [ 0.17525712  4.19183624 13.          0.93093846  1.          0.77496282].
20      [ 1.53502905  8.80147943  5.          0.95765304 16.          0.87842144].
4.571943125342289
```

```
### Bayesian optimization runs (x20): 'exact' Acquisition Function run number = 7

np.random.seed(run_num_7)
surrogate_exact_7 = dGaussianProcess(cov_func, optimize=opt)

X_train7, X_test7, y_train7, y_test7 = train_test_split(X, y, test_size=test_perc, rand

def f_syn_polarity7(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_7, obj
    score = np.array(cross_val_score(reg, X=X_train7, y=y_train7).mean())
    return operator * score

exact_7 = dGPGO(surrogate_exact_7, Acquisition_new(util_exact), f_syn_polarity7, param,
exact_7.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_exact_7 = exact_7.getResult()[0]
params_exact_7['max_depth'] = int(params_exact_7['max_depth'])
params_exact_7['min_child_weight'] = int(params_exact_7['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_exact_train7 = xgb.DMatrix(X_train7, y_train7)
dX_exact_test7 = xgb.DMatrix(X_test7, y_test7)
model_exact_7 = xgb.train(params_exact_7, dX_exact_train7)
pred_exact_7 = model_exact_7.predict(dX_exact_test7)

rmse_exact_7 = np.sqrt(mean_squared_error(pred_exact_7, y_test7))
rmse_exact_7
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [0.76308289 7.79918792 8.         0.98911145 8.         0.98019056].     -0
init     [ 5.3849587   5.01120464 13.          0.74994125  5.          0.88192131].
init     [ 3.30839249  3.9294231  12.          0.6440728  13.          0.41137564].
init     [9.29528191 2.6258377  5.         0.80027446 1.         0.86616513].     -0
init     [ 1.74052764  7.90763512 14.          0.7244129   4.          0.77536887].
1        [3.43305102 3.00339076 8.         0.71322679 4.         0.33322219].     -0
2        [ 6.11582972  9.74457247  9.          0.93736571 17.          0.66419177].
3        [ 5.02225215  1.96894301  6.          0.89946836 15.          0.14043834].
4        [ 8.68686713  7.27037095  5.          0.65538775 10.          0.84203491].
```

```
5       [ 9.19325593  0.04766771 13.         0.95594397 16.         0.80133063].
6       [ 9.9951683   8.29291133 11.         0.98998997 12.         0.32159774].
7       [6.4915356  8.69600226 5.         0.66481282 2.         0.54976375].     -0
8       [ 1.41319611  2.5751638  10.         0.98767908 19.         0.27216707].
9       [9.91546913 0.35662679 7.         0.87706909 8.         0.31009443].     -0
10      [0.24073844 8.35501727 5.         0.58617015 2.         0.49753279].     -0
11      [0.70289771 0.26831186 8.         0.54339287 9.         0.70811191].     -0
12      [ 9.94824081  0.39149062 12.         0.59113076  1.         0.49709751].
13      [ 7.67822637  9.80667081 12.         0.90255729  1.         0.17725725].
14      [ 0.34344431  9.08335969 10.         0.76989135 16.         0.16354482].
15      [ 9.57070021  3.68128181  5.         0.59329325 19.         0.44442317].
16      [ 1.43297985  0.60163245 14.         0.83220406  8.         0.35678245].
17      [ 9.45137179  0.74358028 14.         0.78939199  9.         0.47186777].
18      [ 2.47393117  8.12932833  5.         0.62482376 15.         0.55882726].
19      [ 9.66250325  8.26845592 14.         0.97780715 18.         0.16442988].
20      [8.53226416 4.72551917 6.         0.70114141 6.         0.95505376].     -0
4.499924380514474
```

```
### Bayesian optimization runs (x20): 'exact' Acquisition Function run number = 8

np.random.seed(run_num_8)
surrogate_exact_8 = dGaussianProcess(cov_func, optimize=opt)

X_train8, X_test8, y_train8, y_test8 = train_test_split(X, y, test_size=test_perc, rand

def f_syn_polarity8(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_8, obj
    score = np.array(cross_val_score(reg, X=X_train8, y=y_train8).mean())
    return operator * score

exact_8 = dGPGO(surrogate_exact_8, Acquisition_new(util_exact), f_syn_polarity8, param,
exact_8.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_exact_8 = exact_8.getResult()[0]
params_exact_8['max_depth'] = int(params_exact_8['max_depth'])
params_exact_8['min_child_weight'] = int(params_exact_8['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_exact_train8 = xgb.DMatrix(X_train8, y_train8)
dX_exact_test8 = xgb.DMatrix(X_test8, y_test8)
model_exact_8 = xgb.train(params_exact_8, dX_exact_train8)
pred_exact_8 = model_exact_8.predict(dX_exact_test8)

rmse_exact_8 = np.sqrt(mean_squared_error(pred_exact_8, y_test8))
rmse_exact_8
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 8.73429403  9.68540663 10.         0.68875849  9.         0.48011572].
init     [ 6.12033333  7.66062926  8.         0.76133734 13.         0.93379456].
init     [ 1.46524679  7.01527914  7.         0.90913299 10.         0.36016753].
init     [ 9.73855241  3.33774046 14.         0.53290419  7.         0.7088681 ].
init     [ 3.00618018  1.82702795 11.         0.75681389 14.         0.98627449].
1        [4.42022545 5.48487111 9.         0.97165909 3.         0.63617522].     -0
2        [ 2.28246361  4.04949256 14.         0.80786608  7.         0.81285698].
```

```
3       [ 2.52429836  9.02824683 14.          0.59641093 17.          0.61934886].
4       [ 9.53473907  5.08424998 11.          0.50652828 18.          0.67121466].
5       [6.89072012 1.88822945 5.         0.9252956  8.         0.40577637].     -0
6       [ 2.42575645  9.87357367  5.          0.61143882 19.          0.11833201].
7       [ 6.91772081  1.78286748  5.          0.91502226 18.          0.49192842].
8       [ 3.43855934  0.02041647 14.          0.50910442  1.          0.11324275].
9       [9.69554908 3.70013633 5.         0.72727157 1.         0.49530336].     -0
10      [0.55400363 9.97465254 6.         0.64565646 3.         0.58326638].     -0
11      [1.66316873 0.47469495 7.         0.60820298 1.         0.50689458].     -0
12      [ 1.2277143   1.05411485  5.          0.50198686 13.          0.89258454].
13      [5.63420638 9.7026496  5.         0.64869539 8.         0.22475566].     -0
14      [ 0.63194856  9.08760061 12.          0.51277257  1.          0.85579144].
15      [ 8.35277365  5.17929354 14.          0.67305838  1.          0.70101595].
16      [ 8.62413803  1.63568526 10.          0.78670071 12.          0.5099252 ].
17      [ 3.5121134   5.24088616  9.          0.68973249 19.          0.78159968].
18      [ 2.95370204  9.27299412 14.          0.7994754  10.          0.47804824].
19      [ 8.21638369  0.11131191 14.          0.86289711 17.          0.28011015].
20      [ 7.79664999  9.9555277   7.          0.88786243 19.          0.48083734].
4.462721253378859
```

```
### Bayesian optimization runs (x20): 'exact' Acquisition Function run number = 9

np.random.seed(run_num_9)
surrogate_exact_9 = dGaussianProcess(cov_func, optimize=opt)

X_train9, X_test9, y_train9, y_test9 = train_test_split(X, y, test_size=test_perc, rand

def f_syn_polarity9(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_9, obj
    score = np.array(cross_val_score(reg, X=X_train9, y=y_train9).mean())
    return operator * score

exact_9 = dGPGO(surrogate_exact_9, Acquisition_new(util_exact), f_syn_polarity9, param,
exact_9.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_exact_9 = exact_9.getResult()[0]
params_exact_9['max_depth'] = int(params_exact_9['max_depth'])
params_exact_9['min_child_weight'] = int(params_exact_9['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_exact_train9 = xgb.DMatrix(X_train9, y_train9)
dX_exact_test9 = xgb.DMatrix(X_test9, y_test9)
model_exact_9 = xgb.train(params_exact_9, dX_exact_train9)
pred_exact_9 = model_exact_9.predict(dX_exact_test9)

rmse_exact_9 = np.sqrt(mean_squared_error(pred_exact_9, y_test9))
rmse_exact_9
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 0.10374154  5.01874592 11.          0.50377155  2.          0.29670281].
init     [ 4.18508181  2.48101168 13.          0.69794293  2.          0.25009871].
init     [ 8.78559086  9.50964032 13.          0.98395204 11.          0.90820641].
init     [ 6.66898973  5.47837783  6.          0.97165345 12.          0.72499481].
init     [ 8.24870465  4.65668475 13.          0.68760467  9.          0.98502332].
```

```
1        [6.73714319 2.39608167 5.         0.58130302 3.         0.163077  ].     -0
2        [ 4.24955662  9.67331527 12.          0.64012695  7.          0.96617478].
3        [ 3.60566534  9.79805332 11.          0.62032576 16.          0.3578496 ].
4        [ 4.86601509  0.61279594  8.          0.72162785 18.          0.68911833].
5        [1.01192549e-02 2.87791832e+00 1.30000000e+01 5.02360812e-01
 1.60000000e+01 5.53454779e-01].          -0.4943219873025823    -0.4584168030068045
6        [1.13863488 5.86180669 5.         0.9953054  3.         0.88639082].     -0
7        [9.20185355 9.40235017 8.         0.60433558 1.         0.94540222].     -0
8        [ 7.07313313  8.94084339  5.          0.99439529 19.          0.33798274].
9        [9.55073356 0.68128714 7.         0.82844939 9.         0.80267865].     -0
10       [ 1.48405781  3.2282945  12.          0.72755713  8.          0.45582302].
11       [ 8.41810681  0.28363289 14.          0.70665052 14.          0.98883394].
12       [0.         0.         5.         0.5        6.21322258 0.1        ].     -0
13       [ 8.8503125   5.34876568 11.          0.94536212 19.          0.39914726].
14       [ 1.5822225   4.71560548  5.          0.51218922 16.          0.7972216 ].
15       [4.62485155 9.65182222 5.         0.58896427 8.         0.36022331].     -0
16       [ 8.77945628  5.51694736 12.          0.84613589  4.          0.64741755].
17       [ 3.89448326  9.77793126 11.          0.87312755  1.          0.25420157].
18       [ 4.36796248  0.46487554  9.          0.72990768 12.          0.19754102].
19       [ 2.87706677  6.16957719 14.          0.55246385 12.          0.92866915].
20       [ 1.87277969  9.36196526  9.          0.51233211 11.          0.1932108 ].
4.6575353004618085
```

```
### Bayesian optimization runs (x20): 'exact' Acquisition Function run number = 10

np.random.seed(run_num_10)
surrogate_exact_10 = dGaussianProcess(cov_func, optimize=opt)

X_train10, X_test10, y_train10, y_test10 = train_test_split(X, y, test_size=test_perc,

def f_syn_polarity10(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_10, ob
    score = np.array(cross_val_score(reg, X=X_train10, y=y_train10).mean())
    return operator * score

exact_10 = dGPGO(surrogate_exact_10, Acquisition_new(util_exact), f_syn_polarity10, par
exact_10.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_exact_10 = exact_10.getResult()[0]
params_exact_10['max_depth'] = int(params_exact_10['max_depth'])
params_exact_10['min_child_weight'] = int(params_exact_10['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_exact_train10 = xgb.DMatrix(X_train10, y_train10)
dX_exact_test10 = xgb.DMatrix(X_test10, y_test10)
model_exact_10 = xgb.train(params_exact_10, dX_exact_train10)
pred_exact_10 = model_exact_10.predict(dX_exact_test10)

rmse_exact_10 = np.sqrt(mean_squared_error(pred_exact_10, y_test10))
rmse_exact_10
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 7.71320643  0.20751949  5.          0.72150747 17.          0.12265456].
init     [ 7.0920801   2.65566127 13.          0.57518893 17.          0.83494165].
```

```
init     [ 3.36071584  8.90816531  6.          0.86087766 15.          0.75469196].
init     [ 5.40880931  1.31458152  8.          0.57108502 14.          0.62551123].
init     [1.82631436 8.26082248 6.         0.80888349 5.         0.15900694].     -0
1        [8.31989768 3.09778055 7.         0.64798085 3.         0.98471878].     -0
2        [ 1.51483713  6.46720195 14.          0.87676044  8.          0.10934204].
3        [ 1.40638864  6.36994003 14.          0.94554832 18.          0.42624162].
4        [ 6.23532773  8.31439809 13.          0.65134225  1.          0.75774237].
5        [ 9.32103763  8.04997374 14.          0.76165598 11.          0.58576369].
6        [ 2.70513667  1.83577987 12.          0.56122064  1.          0.15848231].
7        [ 9.16520307  0.72602801 12.          0.91999471  9.          0.54336218].
8        [ 0.13114685  2.69967978  5.          0.89490476 19.          0.55706236].
9        [ 9.57603828  8.81375557  5.          0.56495862 19.          0.45710368].
10       [0.         1.09967587 5.         0.5        9.58020938 0.1       ].     -0
11       [0.40833691 2.05040396 5.         0.83675528 2.         0.93839002].     -0
12       [ 0.06083858  0.83347778 13.          0.60208867 14.          0.81102115].
13       [6.98730996 4.97466872 6.         0.84829188 9.         0.46079499].     -0
14       [ 0.42759101  9.94526216 13.          0.83538077 13.          0.34479326].
15       [9.80244061 9.59221257 6.         0.81318958 4.         0.96637387].     -0
16       [3.30594456 0.27909177 9.         0.73871386 6.         0.22083085].     -0
17       [ 9.71047899  2.77697649 13.          0.98225484  2.          0.32725448].
18       [ 3.45239739  3.71668204  9.          0.77283325 19.          0.7941719 ].
19       [ 8.10151611  9.78827602 14.          0.68048433 19.          0.19392154].
20       [ 9.10340821  9.56875896  8.          0.96862007 13.          0.97727525].
5.142778903321004
```

```
### Bayesian optimization runs (x20): 'exact' Acquisition Function run number = 11

np.random.seed(run_num_11)
surrogate_exact_11 = dGaussianProcess(cov_func, optimize=opt)

X_train11, X_test11, y_train11, y_test11 = train_test_split(X, y, test_size=test_perc,

def f_syn_polarity11(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_11, ob
    score = np.array(cross_val_score(reg, X=X_train11, y=y_train11).mean())
    return operator * score

exact_11 = dGPGO(surrogate_exact_11, Acquisition_new(util_exact), f_syn_polarity11, par
exact_11.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_exact_11 = exact_11.getResult()[0]
params_exact_11['max_depth'] = int(params_exact_11['max_depth'])
params_exact_11['min_child_weight'] = int(params_exact_11['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_exact_train11 = xgb.DMatrix(X_train11, y_train11)
dX_exact_test11 = xgb.DMatrix(X_test11, y_test11)
model_exact_11 = xgb.train(params_exact_11, dX_exact_train11)
pred_exact_11 = model_exact_11.predict(dX_exact_test11)

rmse_exact_11 = np.sqrt(mean_squared_error(pred_exact_11, y_test11))
rmse_exact_11
```

```
    Evaluation       Proposed point           Current eval.          Best eval.
```

```
init     [ 1.80269689  0.19475241  6.          0.59705781 13.          0.47818324].
init     [ 4.85427098  0.12780815  5.          0.91309068 14.          0.86571558].
init     [ 7.2996447   1.08736072 10.          0.92857712 18.          0.66910061].
init     [ 0.20483613  1.16737269  7.          0.57895615 16.          0.83644782].
init     [ 3.44624491  3.18798797 14.          0.54197657 15.          0.63958906].
1        [ 3.00661074  5.9321586  13.          0.93375268  7.          0.32533139].
2        [7.20868744 6.83428849 8.         0.68837575 6.         0.28591593].     -0
3        [ 4.28856064  9.51262941  9.          0.76647308 19.          0.1192808 ].
4        [0.  0.  5.  0.5 1.  0.1].      -0.6979829745079476    -0.4989862321917034
5        [ 6.44177252  1.37885806 12.          0.74571582  1.          0.110379  ].
6        [ 8.80621297  1.41648139 14.          0.58742169  8.          0.82618778].
7        [0.56806539 7.69746805 6.         0.94796222 9.         0.37129588].     -0
8        [0.47065357 6.80536856 5.         0.94482943 1.         0.93677618].     -0
9        [ 5.5178254   9.65002018 13.          0.7661485  13.          0.91577641].
10       [ 9.62795963  5.53773092  5.          0.74613073 18.          0.44845649].
11       [ 8.61765552  0.94349031 10.          0.85641829  4.          0.91576259].
12       [9.1689709  8.67069462 6.         0.8799734  8.         0.25184132].     -0
13       [ 0.25928331  1.26365177 10.          0.55099024  5.          0.58845859].
14       [ 8.40730244  5.3235921  14.          0.57824885  9.          0.96689527].
15       [ 8.27597005  9.04237822 13.          0.54506528  3.          0.15059174].
16       [ 9.7602473   8.08368399 12.          0.65958526 18.          0.9544452 ].
17       [9.96111126 6.75429616 8.         0.78500956 1.         0.98009511].     -0
18       [5.61170222 0.67547973 9.         0.79238564 3.         0.84547685].     -0
19       [ 0.03452186  8.83860607 13.          0.62313645 17.          0.60519619].
20       [ 9.73835913  1.55882873  8.          0.78667186 10.          0.52052364].
4.794073214993642
```

```
### Bayesian optimization runs (x20): 'exact' Acquisition Function run number = 12

np.random.seed(run_num_12)
surrogate_exact_12 = dGaussianProcess(cov_func, optimize=opt)

X_train12, X_test12, y_train12, y_test12 = train_test_split(X, y, test_size=test_perc,

def f_syn_polarity12(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_12, ob
    score = np.array(cross_val_score(reg, X=X_train12, y=y_train12).mean())
    return operator * score

exact_12 = dGPGO(surrogate_exact_12, Acquisition_new(util_exact), f_syn_polarity12, par
exact_12.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_exact_12 = exact_12.getResult()[0]
params_exact_12['max_depth'] = int(params_exact_12['max_depth'])
params_exact_12['min_child_weight'] = int(params_exact_12['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_exact_train12 = xgb.DMatrix(X_train12, y_train12)
dX_exact_test12 = xgb.DMatrix(X_test12, y_test12)
model_exact_12 = xgb.train(params_exact_12, dX_exact_train12)
pred_exact_12 = model_exact_12.predict(dX_exact_test12)

rmse_exact_12 = np.sqrt(mean_squared_error(pred_exact_12, y_test12))
rmse_exact_12
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [1.54162842 7.40049697 6.         0.54321714 4.         0.11311747].     -0
init     [ 9.18747008  9.00714854 14.         0.97847467 11.         0.35544552].
init     [ 6.06083184  9.44225136 14.         0.95626942  5.         0.56910342].
init     [ 5.52037633  4.85377414  7.         0.97886436 17.         0.78810441].
init     [ 0.20809798  1.35210178  5.         0.65494879 16.         0.36062811].
1        [9.46555822 8.57190559 5.         0.50164398 5.         0.71992807].     -0
2        [9.04256367 2.61736915 8.         0.66026854 8.         0.14510453].     -0
3        [ 5.00829307  3.38593972 11.         0.93206268  3.         0.37397626].
4        [ 0.24796255  2.18203944 14.         0.56497025 17.         0.63132662].
5        [ 1.93384153  7.13950146  8.         0.85480597 18.         0.33734734].
6        [ 0.40359854  2.22527636 10.         0.60258213 10.         0.38255809].
7        [ 2.74186895  8.07635126 13.         0.52228241 13.         0.8380668 ].
8        [ 7.63658847  0.39719075 14.         0.96199388 14.         0.84093877].
9        [0.31974577 0.28426423 5.         0.59429506 1.         0.19680364].     -0
10       [ 4.27921374  9.2199845   6.         0.67076861 12.         0.56605459].
11       [7.8127487  4.2013177  5.         0.76692189 1.         0.91258108].     -0
12       [ 9.06259994  1.4172751   5.         0.81347212 19.         0.38881712].
13       [ 9.38461996  5.62749581 12.         0.98689844 17.         0.89710846].
14       [ 4.55964005  0.30434123  5.         0.52399968 11.         0.14256504].
15       [ 6.59943135  1.71178305 14.         0.69283152  8.         0.30459736].
16       [4.73437062 4.4400116  6.         0.99309254 5.         0.34740604].     -0
17       [ 0.96920037  6.62909386 14.         0.51410819  2.         0.40605393].
18       [ 9.28353185  7.64677585 13.         0.64776953  1.         0.57523721].
19       [ 0.74125723  1.19413421 14.         0.98017931  5.         0.4597857 ].
20       [ 9.66945773  0.96666963 10.         0.70845514  1.         0.55906204].
4.772452859101784
```

```
### Bayesian optimization runs (x20): 'exact' Acquisition Function run number = 13

np.random.seed(run_num_13)
surrogate_exact_13 = dGaussianProcess(cov_func, optimize=opt)

X_train13, X_test13, y_train13, y_test13 = train_test_split(X, y, test_size=test_perc,

def f_syn_polarity13(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_13, ob
    score = np.array(cross_val_score(reg, X=X_train13, y=y_train13).mean())
    return operator * score

exact_13 = dGPGO(surrogate_exact_13, Acquisition_new(util_exact), f_syn_polarity13, par
exact_13.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_exact_13 = exact_13.getResult()[0]
params_exact_13['max_depth'] = int(params_exact_13['max_depth'])
params_exact_13['min_child_weight'] = int(params_exact_13['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_exact_train13 = xgb.DMatrix(X_train13, y_train13)
dX_exact_test13 = xgb.DMatrix(X_test13, y_test13)
model_exact_13 = xgb.train(params_exact_13, dX_exact_train13)
pred_exact_13 = model_exact_13.predict(dX_exact_test13)

rmse_exact_13 = np.sqrt(mean_squared_error(pred_exact_13, y_test13))
```

```
rmse_exact_13 = np.sqrt(mean_squared_error(pred_exact_13, y_test13))
rmse_exact_13
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 7.77702411  2.3754122  11.          0.94649135 13.          0.7827256 ].
init     [ 7.51661514  6.07343344 11.          0.69402149 11.          0.13153287].
init     [ 2.98449471  0.58512492 10.          0.73579614 12.          0.33065195].
init     [ 3.47581215  0.0941277  11.          0.86143432  8.          0.58454932].
init     [ 4.70137857  6.24432527 10.          0.8149145  18.          0.10784416].
1        [1.1119361  5.43221306 6.         0.56899303 8.         0.32100319].     -0
2        [5.39023698 3.80105709 8.         0.54170057 1.         0.56798884].     -0
3        [ 0.5185863   5.23876151 13.          0.63798348  5.          0.7914799 ].
4        [ 9.65518672  0.13040633  6.          0.63296628 18.          0.44133279].
5        [ 9.80722669  7.22571611  7.          0.5026908  19.          0.92519376].
6        [ 6.75965929  9.42320667 14.          0.75932127  4.          0.51195623].
7        [9.95671825 0.88335607 5.         0.76593799 7.         0.60049246].     -0
8        [ 1.88898055  9.92199995 14.          0.53783103 12.          0.63359705].
9        [0.32121091 9.03384774 6.         0.79493663 1.         0.29330571].     -0
10       [ 9.64211232  3.05396831 11.          0.80784352  5.          0.67910498].
11       [ 1.60018805  0.54211528  7.          0.80910607 18.          0.55232873].
12       [ 6.1829314   9.53799326  5.          0.66916589 13.          0.57013155].
13       [7.99022981 8.23715587 5.         0.79172469 5.         0.39852784].     -0
14       [ 8.15066897  1.98276445 13.          0.64083395 19.          0.16153982].
15       [ 2.48128047  9.70146472  6.          0.59664322 19.          0.9781653 ].
16       [ 9.08793394  9.83715934 14.          0.60607659 14.          0.5136415 ].
17       [ 1.80857777  4.26197    14.          0.84061579 16.          0.40224246].
18       [ 5.1333583   3.10658752 14.          0.74854361  1.          0.92539511].
19       [ 0.65009462  9.91088996 13.          0.92448902  2.          0.37838494].
20       [0.         0.         5.         0.5        5.17679633 0.1       ].     -0
4.601162573826364
```

```
### Bayesian optimization runs (x20): 'exact' Acquisition Function run number = 14

np.random.seed(run_num_14)
surrogate_exact_14 = dGaussianProcess(cov_func, optimize=opt)

X_train14, X_test14, y_train14, y_test14 = train_test_split(X, y, test_size=test_perc,

def f_syn_polarity14(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_14, ob
    score = np.array(cross_val_score(reg, X=X_train14, y=y_train14).mean())
    return operator * score

exact_14 = dGPGO(surrogate_exact_14, Acquisition_new(util_exact), f_syn_polarity14, par
exact_14.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_exact_14 = exact_14.getResult()[0]
params_exact_14['max_depth'] = int(params_exact_14['max_depth'])
params_exact_14['min_child_weight'] = int(params_exact_14['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_exact_train14 = xgb.DMatrix(X_train14, y_train14)
dX_exact_test14 = xgb.DMatrix(X_test14, y_test14)
model_exact_14 = xgb.train(params_exact_14, dX_exact_train14)
pred_exact_14 = model_exact_14.predict(dX_exact_test14)
```

```
27
28 rmse_exact_14 = np.sqrt(mean_squared_error(pred_exact_14, y_test14))
29 rmse_exact_14
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 5.13943344  7.73165052 12.         0.6831412  11.         0.37876233].
init     [ 9.57603739  5.13116712 14.         0.76959997 12.         0.71328228].
init     [5.34950319 2.47493539 5.         0.50293689 6.         0.29706373].     -0
init     [ 2.94506579  3.45329697  8.         0.87620946 14.         0.9783044 ].
init     [ 1.11811929  1.73004086  5.         0.73745288 12.         0.20586008].
1        [ 6.50637223  2.67617722 14.         0.53562507  1.         0.16862152].
2        [ 9.97732733  0.9008687  13.         0.65397817 19.         0.96533011].
3        [ 6.6877751   9.48200682  5.         0.90861826 11.         0.9411861 ].
4        [9.32373648 9.05676215 9.         0.53064322 3.         0.70657534].     -0
5        [ 1.03495546  6.36597852 14.         0.95388015  5.         0.12529639].
6        [1.11245291 7.32536364 9.         0.80287741 8.         0.8335634 ].     -0
7        [ 0.9687803   2.15143442 14.         0.51651811 18.         0.56051657].
8        [ 5.82553366  9.44318255  6.         0.63592424 19.         0.54190567].
9        [ 8.97462379  3.07976653  6.         0.89115666 18.         0.90399549].
10       [ 2.08494663  9.70581169 14.         0.64307382  3.         0.10613693].
11       [ 8.75151385  7.75347333 14.         0.98873367 11.         0.55773854].
12       [ 5.13735441  7.34260278 13.         0.50594951 19.         0.81583776].
13       [0.  0.  5.  0.5 1.  0.1].       -0.6278549001305299    -0.4448140077853998
14       [3.93459841 8.46767954 5.         0.86837551 3.         0.67826473].     -0
15       [ 0.4418527   4.49961454  5.         0.81022894 19.         0.68885116].
16       [ 2.37733654  0.06526826 14.         0.97689212  7.         0.3543897 ].
17       [ 9.64196863  0.25770476  7.         0.95281681 11.         0.28516842].
18       [9.95902868 1.23916696 7.         0.60336262 2.         0.41962125].     -0
19       [0.88193847 5.43558093 9.         0.70213239 1.         0.9736566 ].     -0
20       [ 8.21075716  2.88666417 12.         0.5197222   7.         0.73645545].
5.080359514778575
```

```
1 ### Bayesian optimization runs (x20): 'exact' Acquisition Function run number = 15
2
3 np.random.seed(run_num_15)
4 surrogate_exact_15 = dGaussianProcess(cov_func, optimize=opt)
5
6 X_train15, X_test15, y_train15, y_test15 = train_test_split(X, y, test_size=test_perc,
7
8 def f_syn_polarity15(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
9     reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
10          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_15, ob
11    score = np.array(cross_val_score(reg, X=X_train15, y=y_train15).mean())
12    return operator * score
13
14 exact_15 = dGPGO(surrogate_exact_15, Acquisition_new(util_exact), f_syn_polarity15, par
15 exact_15.run(max_iter = max_iter, init_evals = n_init) # run
16
17 ### Return optimal parameters' set:
18 params_exact_15 = exact_15.getResult()[0]
19 params_exact_15['max_depth'] = int(params_exact_15['max_depth'])
20 params_exact_15['min_child_weight'] = int(params_exact_15['min_child_weight'])
21
22 ### Re-train with optimal parameters, run predictons:
23 dX_exact_train15 = xgb.DMatrix(X_train15, y_train15)
24 dX_exact_test15 = xgb.DMatrix(X_test15, y_test15)
```

```
model_exact_15 = xgb.train(params_exact_15, dX_exact_train15)
pred_exact_15 = model_exact_15.predict(dX_exact_test15)

rmse_exact_15 = np.sqrt(mean_squared_error(pred_exact_15, y_test15))
rmse_exact_15
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 8.48817697  1.78895925 12.          0.55549316  8.          0.93397854].
init     [ 0.24953032  8.22298097 12.          0.62494951 11.          0.12924598].
init     [ 5.02017228  5.50882771 11.          0.85295832 19.          0.13548008].
init     [2.0023081  9.98543403 7.         0.6295772  2.         0.526127  ].     -0
init     [ 5.09715306  9.45038417 11.          0.7388277  16.          0.22739973].
1        [ 0.29158961  4.9949242  12.          0.89124583  3.          0.67554049].
2        [3.68214008 4.55748717 6.         0.60488381 8.         0.88973248].     -0
3        [9.75991344 6.15203198 6.         0.65490407 1.         0.73816291].     -0
4        [ 9.51793103  9.55070381  6.          0.93315157 11.          0.40416985].
5        [ 6.65116837  8.16324548 14.          0.95750787  1.          0.74925927].
6        [ 0.41861043  0.05076637  5.          0.78940316 12.          0.77619725].
7        [ 0.44758083  1.21361479 14.          0.72053858 16.          0.73785377].
8        [ 9.36080884  1.0313168   5.          0.6330142  14.          0.51274968].
9        [ 3.89599673  7.59173972  5.          0.94509004 14.          0.68417344].
10       [4.3552321  3.34830177 9.         0.76229473 1.         0.13073039].     -0
11       [ 8.34799874  7.25398106 12.          0.79127771  8.          0.66608041].
12       [ 3.91025291  0.94746935  6.          0.83827135 18.          0.68489482].
13       [ 1.41127451  0.02455301 14.          0.99438541  9.          0.56009967].
14       [ 7.09557213  0.02461437 14.          0.76122797 15.          0.64719348].
15       [4.17102171 2.37589806 9.         0.69724056 2.         0.3810416 ].     -0
16       [0.  0.  5.  0.5 1.  0.1].      -0.6979248471795549    -0.4743330993644729
17       [ 9.39083863  5.32748652 10.          0.99453894 14.          0.35726977].
18       [ 7.9436584   8.63416202  5.          0.90235853 19.          0.37180179].
19       [ 1.30779693  9.32518463 14.          0.80733828 19.          0.87810773].
20       [0.01993927 9.65913928 7.         0.55264739 8.         0.94506564].     -0
4.772034603321413
```

```
### Bayesian optimization runs (x20): 'exact' Acquisition Function run number = 16

np.random.seed(run_num_16)
surrogate_exact_16 = dGaussianProcess(cov_func, optimize=opt)

X_train16, X_test16, y_train16, y_test16 = train_test_split(X, y, test_size=test_perc,

def f_syn_polarity16(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_16, ob
    score = np.array(cross_val_score(reg, X=X_train16, y=y_train16).mean())
    return operator * score

exact_16 = dGPGO(surrogate_exact_16, Acquisition_new(util_exact), f_syn_polarity16, par
exact_16.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_exact_16 = exact_16.getResult()[0]
params_exact_16['max_depth'] = int(params_exact_16['max_depth'])
params_exact_16['min_child_weight'] = int(params_exact_16['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_exact_train16 = xgb.DMatrix(X_train16, y_train16)
```

```
dX_exact_train16 = xgb.DMatrix(X_train16, y_train16)
dX_exact_test16 = xgb.DMatrix(X_test16, y_test16)
model_exact_16 = xgb.train(params_exact_16, dX_exact_train16)
pred_exact_16 = model_exact_16.predict(dX_exact_test16)

rmse_exact_16 = np.sqrt(mean_squared_error(pred_exact_16, y_test16))
rmse_exact_16
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [2.23291079 5.23163341 6.         0.65430839 5.         0.30077285].     -0
init     [6.88726162 1.63731425 7.         0.97050543 2.         0.25392012].     -0
init     [ 5.94328983  5.6393473   5.          0.67602695 19.          0.42538144].
init     [ 0.88741148  3.08148142 14.          0.56043938  9.          0.27515386].
init     [ 2.74631586  1.30996118 11.          0.52160786  8.          0.27956463].
1        [ 7.8937256   1.5972923  14.          0.61610774 17.          0.78739284].
2        [ 9.01655783  8.21383177  9.          0.60772965 10.          0.9401803 ].
3        [ 4.35132073  9.89698316 12.          0.94137984 16.          0.57741056].
4        [ 3.38377852  9.31285251 11.          0.88244942  1.          0.67627774].
5        [ 0.02157337  9.97534925  5.          0.75404051 13.          0.40760752].
6        [ 0.19317903  0.6596816   6.          0.86233301 15.          0.41906313].
7        [ 9.2502617   3.00821528  6.          0.66523104 13.          0.86141057].
8        [ 6.47975614  6.0777358  14.          0.94769551 10.          0.71957196].
9        [0.  0.  5.  0.5 1.  0.1].       -0.7101090677178151    -0.4830822737254163
10       [8.12731949 9.23712019 5.         0.71442381 2.         0.11528188].     -0
11       [ 2.41144535  2.21533362 12.          0.76496867  2.          0.96363399].
12       [ 1.24563612  3.10729143 13.          0.58075831 18.          0.47226166].
13       [ 8.07210564  2.37313091 13.          0.95420407  5.          0.85204589].
14       [2.43756281e-03 1.86203502e-01 1.10000000e+01 5.84519618e-01
 1.30000000e+01 2.30149841e-01].          -0.7072748319481598    -0.4574614661469763
15       [ 0.04536026  8.66345072  8.          0.81293119 18.          0.29833721].
16       [ 9.28336661  6.57187836 11.          0.54297007 17.          0.56210295].
17       [ 8.15124198  9.52841363 14.          0.99393976  5.          0.10936576].
18       [ 7.67925358  0.29024289 11.          0.73122052 11.          0.40437158].
19       [ 6.18013749  9.69222015  6.          0.87704216 14.          0.79491401].
20       [ 4.77306317  4.41011512 11.          0.84776341 14.          0.81171734].
4.701236723580356
```

```
### Bayesian optimization runs (x20): 'exact' Acquisition Function run number = 17

np.random.seed(run_num_17)
surrogate_exact_17 = dGaussianProcess(cov_func, optimize=opt)

X_train17, X_test17, y_train17, y_test17 = train_test_split(X, y, test_size=test_perc,

def f_syn_polarity17(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_17, ob
    score = np.array(cross_val_score(reg, X=X_train17, y=y_train17).mean())
    return operator * score

exact_17 = dGPGO(surrogate_exact_17, Acquisition_new(util_exact), f_syn_polarity17, par
exact_17.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_exact_17 = exact_17.getResult()[0]
params_exact_17['max_depth'] = int(params_exact_17['max_depth'])
params_exact_17['min_child_weight'] = int(params_exact_17['min_child_weight'])
```

```
### Re-train with optimal parameters, run predictons:
dX_exact_train17 = xgb.DMatrix(X_train17, y_train17)
dX_exact_test17 = xgb.DMatrix(X_test17, y_test17)
model_exact_17 = xgb.train(params_exact_17, dX_exact_train17)
pred_exact_17 = model_exact_17.predict(dX_exact_test17)

rmse_exact_17 = np.sqrt(mean_squared_error(pred_exact_17, y_test17))
rmse_exact_17
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 2.94665003  5.30586756 11.          0.94443241 14.          0.80828691].
init     [ 6.56333522  6.37520896 12.          0.81487881 18.          0.42203224].
init     [ 9.45683187  0.6004468  11.          0.5171566  10.          0.53881211].
init     [2.72705857 1.19063434 6.         0.74176431 6.         0.10101151].     -0
init     [ 4.77631812  5.24671297 13.          0.66254476 19.          0.36708086].
1        [ 0.65702322  5.79284078 13.          0.75136902  1.          0.30306068].
2        [ 6.93446178  8.68032298 13.          0.78195789  7.          0.91906958].
3        [9.72843652 3.88893279 9.         0.6901555  1.         0.31608219].     -0
4        [ 9.65057736  8.52725784  5.          0.68420234 13.          0.40008732].
5        [ 4.97204887  2.40072226  5.          0.54268748 19.          0.30995407].
6        [0.12174033 8.73496008 5.         0.89827646 5.         0.85354798].     -0
7        [ 2.91443079  0.16723755 13.          0.598201    6.          0.91729605].
8        [7.20615247 9.36901627 6.         0.85465034 7.         0.6878262 ].     -0
9        [ 9.02586164  0.59354638 10.          0.86038693 18.          0.90794111].
10       [ 1.66641474  7.47633023  5.          0.68203645 17.          0.15512069].
11       [ 0.12410542  7.60180472 13.          0.97425003  8.          0.76245421].
12       [0.         0.         8.68066173 0.5        1.         0.1        ].     -0
13       [9.59190042 1.26233772 5.         0.55398722 6.         0.46259714].     -0
14       [ 9.06002681  9.03779351 14.          0.60614154  1.          0.16421461].
15       [ 0.91894312  0.          5.          0.5        14.21833141  0.1        ].
16       [ 8.69529605  0.27226865 14.          0.90636352  1.          0.39638219].
17       [ 4.56335883  2.35318071 11.          0.75925047 16.          0.7697429 ].
18       [ 0.25691043  8.80430419 11.          0.61482757 19.          0.833227  ].
19       [ 9.24004771  6.36004976 11.          0.81644504 12.          0.66235401].
20       [ 7.9822505   0.1930237   5.          0.88184327 12.          0.30139498].
4.758964442352727
```

```
### Bayesian optimization runs (x20): 'exact' Acquisition Function run number = 18

np.random.seed(run_num_18)
surrogate_exact_18 = dGaussianProcess(cov_func, optimize=opt)

X_train18, X_test18, y_train18, y_test18 = train_test_split(X, y, test_size=test_perc,

def f_syn_polarity18(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_18, ob
    score = np.array(cross_val_score(reg, X=X_train18, y=y_train18).mean())
    return operator * score

exact_18 = dGPGO(surrogate_exact_18, Acquisition_new(util_exact), f_syn_polarity18, par
exact_18.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_exact_18 = exact_18.getResult()[0]
params_exact_18['max_depth'] = int(params_exact_18['max_depth'])
```

```
params_exact_18['max_depth'] = int(params_exact_18['max_depth'])
params_exact_18['min_child_weight'] = int(params_exact_18['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_exact_train18 = xgb.DMatrix(X_train18, y_train18)
dX_exact_test18 = xgb.DMatrix(X_test18, y_test18)
model_exact_18 = xgb.train(params_exact_18, dX_exact_train18)
pred_exact_18 = model_exact_18.predict(dX_exact_test18)

rmse_exact_18 = np.sqrt(mean_squared_error(pred_exact_18, y_test18))
rmse_exact_18
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [6.50374242 5.05453374 6.         0.59092011 3.         0.28357516].     -0
init     [0.11506734 4.26891483 9.         0.81785956 5.         0.63489043].     -0
init     [ 2.8861259   6.35547834 11.          0.64267955 14.          0.27877092].
init     [6.57189031 6.99655629 8.         0.63235896 4.         0.52894035].     -0
init     [ 6.66600348  2.11312037 14.          0.74363461  4.          0.73174558].
1        [ 8.67093232  0.11649132  5.          0.92962202 15.          0.53672863].
2        [ 8.43851229  2.41114508 13.          0.75771586 19.          0.86905071].
3        [ 9.44281001  9.01534322  7.          0.99142432 16.          0.37631199].
4        [ 3.19538294  9.91737336 14.          0.7976317   5.          0.25704487].
5        [ 1.97643014  8.37982471  5.          0.63246176 17.          0.45403539].
6        [ 1.26601315  0.31299408  6.          0.95296222 16.          0.13649883].
7        [ 1.18347798  2.03195078 14.          0.62549517 10.          0.6517083 ].
8        [6.72039962 1.0287777  9.         0.62848622 9.         0.70815446].     -0
9        [ 7.49192948  9.60283663 14.          0.93718151 19.          0.24625933].
10       [ 3.63870552  9.73349763  7.          0.73625258 11.          0.87968363].
11       [0.         0.         5.90250308 0.5        1.         0.1        ].     -0
12       [ 1.68019344  0.20490035 13.          0.89332378 19.          0.17761947].
13       [ 9.38974126  5.86345962 14.          0.7410326  12.          0.80551421].
14       [ 0.81446253  9.28654597 10.          0.69058086  1.          0.88113937].
15       [0.         0.         5.         0.5        8.29200619 0.1        ].     -0
16       [ 7.5232407   0.14021923 14.          0.63702182 13.          0.19869251].
17       [9.18740115 8.8684293  7.         0.964289   9.         0.7517104 ].     -0
18       [ 9.56072091  9.1368036  14.          0.68050527  1.          0.47597207].
19       [ 6.33394188  3.31142716  8.          0.64110795 18.          0.66631055].
20       [ 1.47346798  3.42280397 13.          0.87173303  1.          0.71719115].
4.746067780922308
```

```
### Bayesian optimization runs (x20): 'exact' Acquisition Function run number = 19

np.random.seed(run_num_19)
surrogate_exact_19 = dGaussianProcess(cov_func, optimize=opt)

X_train19, X_test19, y_train19, y_test19 = train_test_split(X, y, test_size=test_perc,

def f_syn_polarity19(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_19, ob
    score = np.array(cross_val_score(reg, X=X_train19, y=y_train19).mean())
    return operator * score

exact_19 = dGPGO(surrogate_exact_19, Acquisition_new(util_exact), f_syn_polarity19, par
exact_19.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
```

```
params_exact_19 = exact_19.getResult()[0]
params_exact_19['max_depth'] = int(params_exact_19['max_depth'])
params_exact_19['min_child_weight'] = int(params_exact_19['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_exact_train19 = xgb.DMatrix(X_train19, y_train19)
dX_exact_test19 = xgb.DMatrix(X_test19, y_test19)
model_exact_19 = xgb.train(params_exact_19, dX_exact_train19)
pred_exact_19 = model_exact_19.predict(dX_exact_test19)

rmse_exact_19 = np.sqrt(mean_squared_error(pred_exact_19, y_test19))
rmse_exact_19
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 0.97533602  7.61249717 13.          0.85765469 11.          0.39830191].
init     [ 0.82999565  6.71977081  6.          0.50407413 19.          0.67209466].
init     [ 2.15923256  5.49027432 12.          0.52588686 10.          0.20235326].
init     [4.99659267 1.52108422 6.         0.73481085 4.         0.71949465].     -0
init     [ 3.72927156  9.46160045  5.          0.80554614 18.          0.97708466].
1        [ 8.33060043  1.42030563  8.          0.92863724 14.          0.78606141].
2        [ 4.70068371  4.9755295  14.          0.98901029  1.          0.96039614].
3        [5.72592037 9.8573523  7.         0.62344236 5.         0.18058155].     -0
4        [ 3.65000245  2.90359952 13.          0.98940034 19.          0.29019455].
5        [ 9.25987473  8.07672103 11.          0.53393029 18.          0.59858945].
6        [ 9.76186054  4.90300619 10.          0.67721911  6.          0.88084199].
7        [ 1.86342862  0.38142632  7.          0.62393245 10.          0.46180447].
8        [1.60895472e-01 8.27074864e-03 1.30000000e+01 6.91893018e-01
 5.00000000e+00 4.60327119e-01].          -0.5676975299111877    -0.454763872590455
9        [ 0.32873959  7.7155114   5.          0.56871173 11.          0.34471029].
10       [ 6.55489773  0.06438745 14.          0.53351105 11.          0.76391016].
11       [ 6.96994971  9.387078    8.          0.7367405  12.          0.10249646].
12       [ 0.77535612  8.78405201 10.          0.78862322  1.          0.52634822].
13       [ 3.86326609  1.40312943  6.          0.7852563  18.          0.23797267].
14       [ 8.38895111  9.39819527 14.          0.57861776  4.          0.75378716].
15       [ 0.11219244  8.71714366 12.          0.84271463 17.          0.71017966].
16       [0.        0.        5.        0.5       1.1548736 0.1      ].           -0
17       [ 9.89996921  0.59570014 13.          0.67906413  1.          0.37793763].
18       [ 9.4335981   8.77998409 14.          0.6133824  11.          0.10869753].
19       [6.1285861  5.61245367 5.         0.51193162 9.         0.99608349].     -0
20       [1.56098356 6.50826155 8.         0.88730571 6.         0.9791372 ].     -0
4.63424117920896
```

```
### Bayesian optimization runs (x20): 'exact' Acquisition Function run number = 20

np.random.seed(run_num_20)
surrogate_exact_20 = dGaussianProcess(cov_func, optimize=opt)

X_train20, X_test20, y_train20, y_test20 = train_test_split(X, y, test_size=test_perc,

def f_syn_polarity20(alpha, gamma, max_depth, subsample, min_child_weight, colsample):
    reg = XGBRegressor(reg_alpha=alpha, gamma=gamma, max_depth=int(max_depth), subsampl
          colsample_bytree=colsample, n_estimators = n_est, random_state=run_num_20, ob
    score = np.array(cross_val_score(reg, X=X_train20, y=y_train20).mean())
    return operator * score

exact_20 = dGPGO(surrogate_exact_20, Acquisition_new(util_exact), f_syn_polarity20, par
```

```
exact_20.run(max_iter = max_iter, init_evals = n_init) # run

### Return optimal parameters' set:
params_exact_20 = exact_20.getResult()[0]
params_exact_20['max_depth'] = int(params_exact_20['max_depth'])
params_exact_20['min_child_weight'] = int(params_exact_20['min_child_weight'])

### Re-train with optimal parameters, run predictons:
dX_exact_train20 = xgb.DMatrix(X_train20, y_train20)
dX_exact_test20 = xgb.DMatrix(X_test20, y_test20)
model_exact_20 = xgb.train(params_exact_20, dX_exact_train20)
pred_exact_20 = model_exact_20.predict(dX_exact_test20)

rmse_exact_20 = np.sqrt(mean_squared_error(pred_exact_20, y_test20))
rmse_exact_20
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [ 5.88130801  8.97713728 14.         0.81074445  8.         0.95540649].
init     [6.72865655 0.41173329 8.         0.6361582  7.         0.76174061].     -0
init     [ 4.77387703  8.66202323 10.         0.51833215  7.         0.10123387].
init     [ 5.75489985  4.74524381  8.         0.78084343 15.         0.26643049].
init     [ 4.53444     4.47342833  8.         0.91974896 18.         0.35997552].
1        [ 7.96566073  7.15509535  7.         0.79906691 11.         0.34132075].
2        [ 1.98667885  1.35773177 13.         0.57199118  2.         0.39498908].
3        [ 3.00704909  2.42524876 14.         0.95062509 15.         0.83595087].
4        [2.60714073 4.21905962 5.         0.5        2.43949111 0.1       ].     -0
5        [ 9.60625049  8.19354987 13.         0.6912874  18.         0.13826635].
6        [ 8.79913956  5.84239608 13.         0.98638201  1.         0.70495849].
7        [ 0.72788527  2.26655356 10.         0.97273032 15.         0.85843758].
8        [4.62702633 9.6876243  5.         0.9923406  1.         0.38132515].     -0
9        [ 1.44692101  9.96202174 12.         0.76092884 18.         0.10523817].
10       [ 1.01814405  9.81807131 14.         0.52908047  1.         0.89345697].
11       [9.53736401 9.25093671 6.         0.95646227 5.         0.53751962].     -0
12       [ 9.95790482  0.39582549 14.         0.73682131 13.         0.88500223].
13       [7.12982753 6.84003365 7.         0.76458922 2.         0.82832765].     -0
14       [ 7.47036597  0.87634126 13.         0.90366023 19.         0.17017133].
15       [ 5.37444991  0.3056877   9.         0.99777328 15.         0.5903115 ].
16       [0.         0.         5.99088653 0.5        9.99088653 0.1       ].     -0
17       [ 0.0269108   5.63927067 14.         0.80391219 10.         0.28928667].
18       [ 6.16018658  2.45414677 14.         0.93866062  8.         0.74546669].
19       [ 9.64898386  6.20494423  6.         0.8469719  19.         0.57533849].
20       [9.07371937 0.79321581 6.         0.88798724 2.         0.43366394].     -0
4.454763173494824
```

```
end_exact = time.time()
end_exact

time_exact = end_exact - start_exact
time_exact
```

```
685.1470732688904
```

```
rmse_approx = [rmse_approx_1,
rmse_approx_2,
rmse_approx_3,
rmse_approx_4,
```

```
 5 rmse_approx_5,
 6 rmse_approx_6,
 7 rmse_approx_7,
 8 rmse_approx_8,
 9 rmse_approx_9,
10 rmse_approx_10,
11 rmse_approx_11,
12 rmse_approx_12,
13 rmse_approx_13,
14 rmse_approx_14,
15 rmse_approx_15,
16 rmse_approx_16,
17 rmse_approx_17,
18 rmse_approx_18,
19 rmse_approx_19,
20 rmse_approx_20]
21
22 np.mean(rmse_approx)
```

```
4.712012294225476
```

```
 1 rmse_exact = [rmse_exact_1,
 2 rmse_exact_2,
 3 rmse_exact_3,
 4 rmse_exact_4,
 5 rmse_exact_5,
 6 rmse_exact_6,
 7 rmse_exact_7,
 8 rmse_exact_8,
 9 rmse_exact_9,
10 rmse_exact_10,
11 rmse_exact_11,
12 rmse_exact_12,
13 rmse_exact_13,
14 rmse_exact_14,
15 rmse_exact_15,
16 rmse_exact_16,
17 rmse_exact_17,
18 rmse_exact_18,
19 rmse_exact_19,
20 rmse_exact_20]
21
22 np.mean(rmse_exact)
```

```
4.720981224272282
```

```
1 min_rmse_approx = min_max_array(rmse_approx)
2 min_rmse_approx, len(min_rmse_approx)
```

```
([4.667222167746296,
  4.600295852493361,
  4.600295852493361,
  4.600295852493361,
  4.600295852493361,
```

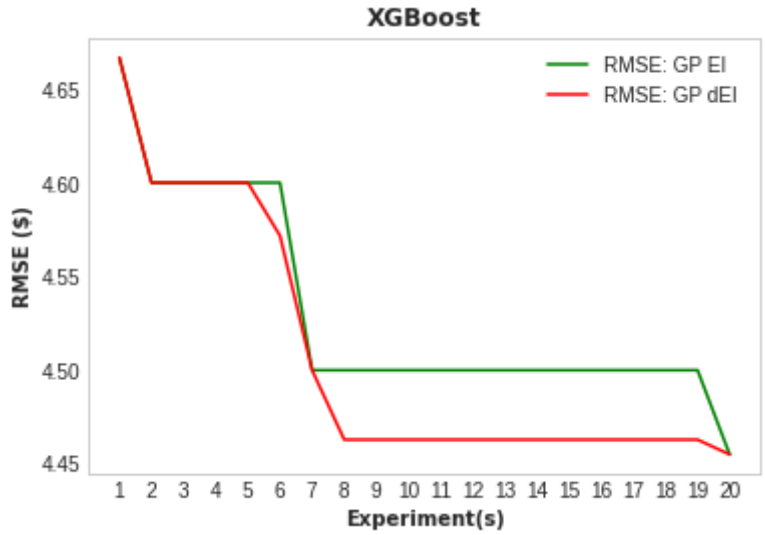
```
 4.600295852493361,
 4.499924380514474,
 4.499924380514474,
 4.499924380514474,
 4.499924380514474,
 4.499924380514474,
 4.499924380514474,
 4.499924380514474,
 4.499924380514474,
 4.499924380514474,
 4.499924380514474,
 4.499924380514474,
 4.499924380514474,
 4.499924380514474,
 4.499924380514474,
 4.454763173494824],
20)
```

```
1 min_rmse_exact = min_max_array(rmse_exact)
2 min_rmse_exact, len(min_rmse_exact)
```

```
([4.667222167746296,
  4.600295852493361,
  4.600295852493361,
  4.600295852493361,
  4.600295852493361,
  4.571943125342289,
  4.499924380514474,
  4.462721253378859,
  4.462721253378859,
  4.462721253378859,
  4.462721253378859,
  4.462721253378859,
  4.462721253378859,
  4.462721253378859,
  4.462721253378859,
  4.462721253378859,
  4.462721253378859,
  4.462721253378859,
  4.462721253378859,
  4.454763173494824],
 20)
```

```
1 ### Visualise!
2
3 title = obj_func
4
5 plt.figure()
6
7 plt.plot(min_rmse_approx, color = 'Green', label='RMSE: GP EI ')
8 plt.plot(min_rmse_exact, color = 'Red', label='RMSE: GP dEI ')# r'($\nu$' ' = {})'.form
9
10 plt.title(title, weight = 'bold', family = 'Arial')
11 plt.xlabel('Experiment(s)', weight = 'bold', family = 'Arial') # x-axis label
12 plt.ylabel('RMSE ($)', weight = 'bold', family = 'Arial') # y-axis label
13 plt.legend(loc=0) # add plot legend
14
15 ### Make the x-ticks integers, not floats:
16 count = len(min_rmse_approx)
```

```
plt.xticks(np.arange(count), np.arange(1, count + 1))
plt.grid(b=None)
plt.show() #visualize!
```




```
time_approx, time_exact
```

```
(802.8092265129089, 685.1470732688904)
```

```
1
```