

Hyperparameter Tuning Cookbook

A guide for scikit-learn, PyTorch, river, and spotPython

Thomas Bartz-Beielstein

Jul 3, 2023

Table of contents

Preface	3
Citation	3
1 Introduction: Hyperparameter Tuning	5
1.1 The Hyperparameter Tuning Software SPOT	6
1.2 Spot as an Optimizer	7
1.3 Example: <code>Spot</code> and the Sphere Function	8
1.3.1 The Objective Function: Sphere	8
1.4 Spot Parameters: <code>fun_evals</code> , <code>init_size</code> and <code>show_models</code>	10
1.5 Print the Results	12
1.6 Show the Progress	12
2 Multi-dimensional Functions	14
2.1 Example: <code>Spot</code> and the 3-dim Sphere Function	14
2.1.1 The Objective Function: 3-dim Sphere	14
2.1.2 Results	15
2.1.3 A Contour Plot	16
2.2 Conclusion	18
2.3 Exercises	18
2.3.1 The Three Dimensional <code>fun_cubed</code>	18
2.3.2 The Ten Dimensional <code>fun_wing_wt</code>	19
2.3.3 The Three Dimensional <code>fun_runge</code>	19
2.3.4 The Three Dimensional <code>fun_linear</code>	19
3 Isotropic and Anisotropic Kriging	20
3.1 Example: Isotropic <code>Spot</code> Surrogate and the 2-dim Sphere Function	20
3.1.1 The Objective Function: 2-dim Sphere	20
3.1.2 Results	21
3.2 Example With Anisotropic Kriging	21
3.2.1 Taking a Look at the <code>theta</code> Values	22
3.3 Exercises	23
3.3.1 <code>fun_branin</code>	23
3.3.2 <code>fun_sin_cos</code>	24
3.3.3 <code>fun_runge</code>	24
3.3.4 <code>fun_wingwt</code>	24

4	Using sklearn Surrogates in spotPython	25
4.1	Example: Branin Function with spotPython's Internal Kriging Surrogate . . .	25
4.1.1	The Objective Function Branin	25
4.1.2	Running the surrogate model based optimizer Spot:	26
4.1.3	Print the Results	26
4.1.4	Show the Progress and the Surrogate	26
4.2	Example: Using Surrogates From scikit-learn	27
4.2.1	GaussianProcessRegressor as a Surrogate	28
4.3	Example: One-dimensional Sphere Function With spotPython's Kriging	30
4.3.1	Results	35
4.4	Example: Sklearn Model GaussianProcess	36
4.5	Exercises	42
4.5.1	DecisionTreeRegressor	42
4.5.2	RandomForestRegressor	42
4.5.3	linear_model.LinearRegression	42
4.5.4	linear_model.Ridge	43
4.6	Exercise 2	43
5	Sequential Parameter Optimization: Using scipy Optimizers	44
5.1	The Objective Function Branin	44
5.2	The Optimizer	45
5.3	Print the Results	46
5.4	Show the Progress	46
5.5	Exercises	47
5.5.1	dual_annealing	47
5.5.2	direct	47
5.5.3	shgo	48
5.5.4	basinhopping	48
5.5.5	Performance Comparison	48
6	Sequential Parameter Optimization: Gaussian Process Models	49
6.1	Gaussian Processes Regression: Basic Introductory scikit-learn Example . .	49
6.1.1	Train and Test Data	50
6.1.2	Building the Surrogate With Sklearn	50
6.1.3	Plotting the SklearnModel	50
6.1.4	The spotPython Version	51
6.1.5	Visualizing the Differences Between the spotPython and the sklearn Model Fits	52
6.2	Exercises	53
6.2.1	Schonlau Example Function	53
6.2.2	Forrester Example Function	53
6.2.3	fun_runge Function (1-dim)	54
6.2.4	fun_cubed (1-dim)	55

6.2.5	The Effect of Noise	55
7	Expected Improvement	57
7.1	Example: <code>Spot</code> and the 1-dim Sphere Function	57
7.1.1	The Objective Function: 1-dim Sphere	57
7.1.2	Results	58
7.2	Same, but with EI as <code>infill_criterion</code>	58
7.3	Non-isotropic Kriging	59
7.4	Using <code>sklearn</code> Surrogates	61
7.4.1	The <code>spot</code> Loop	61
7.4.2	<code>spot</code> : The Initial Model	63
7.4.3	Init: Build Initial Design	63
7.4.4	Evaluate	66
7.4.5	Build Surrogate	66
7.4.6	A Simple Predictor	66
7.5	Gaussian Processes regression: basic introductory example	66
7.6	The Surrogate: Using scikit-learn models	69
7.7	Additional Examples	71
7.7.1	Optimize on Surrogate	75
7.7.2	Evaluate on Real Objective	75
7.7.3	Impute / Infill new Points	75
7.8	Tests	75
7.9	EI: The Famous Schonlau Example	76
7.10	EI: The Forrester Example	78
7.11	Noise	81
7.12	Cubic Function	84
7.13	Factors	90
8	Hyperparameter Tuning and Noise	92
8.1	Example: <code>Spot</code> and the Noisy Sphere Function	92
8.1.1	The Objective Function: Noisy Sphere	92
8.2	Print the Results	96
8.3	Noise and Surrogates: The Nugget Effect	96
8.3.1	The Noisy Sphere	96
8.4	Exercises	99
8.4.1	Noisy <code>fun_cubed</code>	99
8.4.2	<code>fun_runge</code>	100
8.4.3	<code>fun_forrester</code>	100
8.4.4	<code>fun_xsin</code>	100
9	Handling Noise: Optimal Computational Budget Allocation in <code>Spot</code>	101
9.1	Example: <code>Spot</code> , OCBA, and the Noisy Sphere Function	101
9.1.1	The Objective Function: Noisy Sphere	101

9.2	Print the Results	111
9.3	Noise and Surrogates: The Nugget Effect	112
9.3.1	The Noisy Sphere	112
9.4	Exercises	115
9.4.1	Noisy <code>fun_cubed</code>	115
9.4.2	<code>fun_runge</code>	115
9.4.3	<code>fun_forrester</code>	115
9.4.4	<code>fun_xsin</code>	116
10	HPT: sklearn SVC on Moons Data	117
10.1	Step 1: Setup	117
10.2	Step 2: Initialization of the Empty <code>fun_control</code> Dictionary	118
10.3	Step 3: SKlearn Load Data (Classification)	118
10.4	Step 4: Specification of the Preprocessing Model	120
10.5	Step 5: Select Model (<code>algorithm</code>) and <code>core_model_hyper_dict</code>	121
10.6	Step 6: Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	123
10.6.1	Modify hyperparameter of type numeric and integer (boolean)	124
10.6.2	Modify hyperparameter of type factor	124
10.6.3	Optimizers	124
10.7	Step 7: Selection of the Objective (Loss) Function	125
10.7.1	Predict Classes or Class Probabilities	125
10.8	Step 8: Calling the SPOT Function	125
10.8.1	Preparing the SPOT Call	125
10.8.2	The Objective Function	126
10.8.3	Run the <code>Spot</code> Optimizer	126
10.8.4	Starting the Hyperparameter Tuning	127
10.9	Step 9: Results	128
10.9.1	Show variable importance	130
10.9.2	Get Default Hyperparameters	130
10.9.3	Get SPOT Results	131
10.9.4	Plot: Compare Predictions	132
10.9.5	Detailed Hyperparameter Plots	134
10.9.6	Parallel Coordinates Plot	137
10.9.7	Plot all Combinations of Hyperparameters	137
11	HPT: PyTorch With fashionMNIST	138
11.1	Step 1: Setup	138
11.2	Step 2: Initialization of the Empty <code>fun_control</code> Dictionary	140
11.3	Step 3: PyTorch Data Loading	140
11.3.1	Load fashionMNIST Data	140
11.4	Step 4: Specification of the Preprocessing Model	141

11.5	Step 5: Select Model (<code>algorithm</code>) and <code>core_model_hyper_dict</code>	141
11.5.1	The Search Space	142
11.5.2	Configuring the Search Space With <code>spotPython</code>	142
11.6	Step 6: Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	144
11.6.1	Modify hyperparameter of type numeric and integer (boolean)	144
11.6.2	Modify hyperparameter of type factor	145
11.6.3	Optimizers	145
11.7	Step 7: Selection of the Objective (Loss) Function	145
11.7.1	Evaluation	145
11.7.2	Metric	146
11.8	Step 8: Calling the SPOT Function	146
11.8.1	Preparing the SPOT Call	146
11.8.2	The Objective Function <code>fun_torch</code>	147
11.8.3	Starting the Hyperparameter Tuning	147
11.9	Step 9: Tensorboard	152
11.10	Step 10: Results	152
11.10.1	Show variable importance	153
11.10.2	Get the Tuned Architecture (SPOT Results)	154
11.10.3	Get Default Hyperparameters	155
11.10.4	Evaluation of the Default and the Tuned Architectures	155
11.10.5	Detailed Hyperparameter Plots	158
11.10.6	Parallel Coordinates Plot	160
11.10.7	Plot all Combinations of Hyperparameters	160
12	HPT: PyTorch With <code>cifar10</code> Data	161
12.1	Step 1: Setup	161
12.2	Step 2: Initialization of the Empty <code>fun_control</code> Dictionary	163
12.3	Step 3: PyTorch Data Loading	163
12.3.1	Load Data <code>Cifar10</code> Data	163
12.4	Step 4: Specification of the Preprocessing Model	164
12.5	Step 5: Select Model (<code>algorithm</code>) and <code>core_model_hyper_dict</code>	164
12.5.1	Implementing a Configurable Neural Network With <code>spotPython</code>	164
12.5.2	The Search Space	165
12.5.3	Configuring the Search Space With <code>spotPython</code>	165
12.6	Step 6: Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	166
12.6.1	Step 5: Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	167
12.6.2	Modify hyperparameter of type factor	167
12.6.3	Optimizers	167
12.7	Step 7: Selection of the Objective (Loss) Function	168
12.7.1	Evaluation	168

12.7.2	Metric	168
12.8	Step 8: Calling the SPOT Function	169
12.8.1	Preparing the SPOT Call	169
12.8.2	The Objective Function <code>fun_torch</code>	169
12.8.3	Starting the Hyperparameter Tuning	170
12.9	Step 9: Tensorboard	174
12.10	Step 10: Results	174
12.10.1	Show variable importance	176
12.10.2	Get the Tuned Architecture (SPOT Results)	176
12.10.3	Evaluation of the Tuned Architecture	177
12.10.4	Cross-validated Evaluations	178
12.10.5	Detailed Hyperparameter Plots	179
12.10.6	Parallel Coordinates Plot	180
12.10.7	Plot all Combinations of Hyperparameters	180
13	HPT: River	181
13.1	Step 1: Setup	181
13.1.1	<code>river</code> Hyperparameter Tuning: HATR with Friedman Drift Data	181
13.2	Step 2: Initialization of the <code>fun_control</code> Dictionary	182
13.3	Step 3: Load the Friedman Drift Data	182
13.4	Step 4: Specification of the Preprocessing Model	183
13.5	Step 5: Select <code>algorithm</code> and <code>core_model_hyper_dict</code>	184
13.6	Step 6: Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	187
13.6.1	Modify hyperparameter of type factor	187
13.6.2	Modify hyperparameter of type numeric and integer (boolean)	187
13.7	Step 7: Selection of the Objective (Loss) Function	187
13.8	Step 8: Calling the SPOT Function	188
13.8.1	Prepare the SPOT Parameters	188
13.8.2	Run the <code>Spot</code> Optimizer	189
13.9	Step 9: Results	190
13.9.1	Show variable importance	192
13.9.2	Build and Evaluate HTR Model with Tuned Hyperparameters	192
13.9.3	The Large Data Set (k=0.2)	193
13.9.4	Get Default Hyperparameters	194
13.9.5	Get SPOT Results	197
13.9.6	Visualize Regression Trees	201
13.9.7	Spot Model	201
13.9.8	Detailed Hyperparameter Plots	203
13.9.9	Parallel Coordinates Plots	203
13.9.10	Plot all Combinations of Hyperparameters	203

14 HPT: PyTorch With spotPython and Ray Tune on CIFAR10	205
14.1 Step 1: Setup	206
14.2 Step 2: Initialization of the <code>fun_control</code> Dictionary	207
14.3 Step 3: PyTorch Data Loading	208
14.4 Step 4: Specification of the Preprocessing Model	208
14.5 Step 5: Select Model (<code>algorithm</code>) and <code>core_model_hyper_dict</code>	209
14.5.1 The <code>Net_Core</code> class	211
14.5.2 Comparison of the Approach Described in the PyTorch Tutorial With spotPython	211
14.5.3 The Search Space: Hyperparameters	212
14.5.4 Configuring the Search Space With Ray Tune	212
14.5.5 Configuring the Search Space With spotPython	213
14.6 Step 6: Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	215
14.6.1 Optimizers	216
14.7 Step 7: Selection of the Objective (Loss) Function	218
14.7.1 Evaluation: Data Splitting	218
14.7.2 Hold-out Data Split	218
14.7.3 Cross-Validation	219
14.7.4 Overview of the Evaluation Settings	220
14.7.5 Evaluation: Loss Functions and Metrics	221
14.8 Step 8: Calling the SPOT Function	222
14.8.1 Preparing the SPOT Call	222
14.8.2 The Objective Function <code>fun_torch</code>	223
14.8.3 Using Default Hyperparameters or Results from Previous Runs	223
14.8.4 Starting the Hyperparameter Tuning	223
14.9 Step 9: Tensorboard	233
14.9.1 Tensorboard: Start Tensorboard	234
14.9.2 Saving the State of the Notebook	235
14.10 Step 10: Results	235
14.10.1 Get the Tuned Architecture (SPOT Results)	237
14.10.2 Get Default Hyperparameters	238
14.10.3 Evaluation of the Default Architecture	238
14.10.4 Evaluation of the Tuned Architecture	240
14.10.5 Detailed Hyperparameter Plots	242
14.11 Summary and Outlook	244
14.12 Appendix	244
14.12.1 Sample Output From Ray Tune's Run	244
15 HPT: sklearn RandomForestClassifier VBDP Data	246
15.1 Step 1: Setup	246
15.2 Step 2: Initialization of the Empty <code>fun_control</code> Dictionary	247

15.3	Step 3: PyTorch Data Loading	248
15.3.1	Load Data: Classification VBDP	248
15.3.2	Holdout Train and Test Data	248
15.4	Step 4: Specification of the Preprocessing Model	249
15.5	Step 5: Select Model (<code>algorithm</code>) and <code>core_model_hyper_dict</code>	250
15.6	Step 6: Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	252
15.6.1	Modify hyperparameter of type numeric and integer (boolean)	252
15.6.2	Modify hyperparameter of type factor	252
15.6.3	Optimizers	253
15.6.4	Selection of the Objective: Metric and Loss Functions	253
15.7	Step 7: Selection of the Objective (Loss) Function	253
15.7.1	Metric Function	253
15.7.2	Evaluation on Hold-out Data	254
15.7.3	OOB Score	255
15.8	Step 8: Calling the SPOT Function	256
15.8.1	Preparing the SPOT Call	256
15.8.2	The Objective Function	256
15.8.3	Run the <code>Spot</code> Optimizer	257
15.9	Step 9: Tensorboard	259
15.10	Step 10: Results	260
15.10.1	Show variable importance	261
15.10.2	Get Default Hyperparameters	261
15.10.3	Get SPOT Results	262
15.10.4	Evaluate SPOT Results	263
15.10.5	Handling Non-deterministic Results	264
15.10.6	Evaluation of the Default Hyperparameters	264
15.10.7	Plot: Compare Predictions	265
15.10.8	Cross-validated Evaluations	267
15.10.9	Detailed Hyperparameter Plots	268
15.10.10	Parallel Coordinates Plot	276
15.10.11	Plot all Combinations of Hyperparameters	276
16	HPT: sklearn XGB Classifier VBDP Data	277
16.1	Step 1: Setup	277
16.2	Step 2: Initialization of the Empty <code>fun_control</code> Dictionary	278
16.3	Step 3: PyTorch Data Loading	279
16.3.1	1. Load Data: Classification VBDP	279
16.3.2	Holdout Train and Test Data	279
16.4	Step 4: Specification of the Preprocessing Model	280
16.5	Step 5: Select Model (<code>algorithm</code>) and <code>core_model_hyper_dict</code>	281

16.6	Step 6: Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	283
16.6.1	Modify hyperparameter of type numeric and integer (boolean)	283
16.6.2	Modify hyperparameter of type factor	283
16.6.3	Optimizers	284
16.7	Step 7: Selection of the Objective (Loss) Function	284
16.7.1	Evaluation	284
16.7.2	Selection of the Objective: Metric and Loss Functions	284
16.7.3	Loss Function	284
16.7.4	Metric Function	284
16.7.5	Evaluation on Hold-out Data	286
16.8	Step 8: Calling the SPOT Function	286
16.8.1	Preparing the SPOT Call	286
16.8.2	The Objective Function	287
16.8.3	Run the <code>Spot</code> Optimizer	287
16.9	Step 9: Tensorboard	289
16.10	Step 10: Results	289
16.10.1	Show variable importance	291
16.10.2	Get Default Hyperparameters	291
16.10.3	Get SPOT Results	292
16.10.4	Evaluate SPOT Results	293
16.10.5	Handling Non-deterministic Results	294
16.10.6	Evaluation of the Default Hyperparameters	294
16.10.7	Plot: Compare Predictions	295
16.10.8	Cross-validated Evaluations	297
16.10.9	Detailed Hyperparameter Plots	298
16.10.10	Parallel Coordinates Plot	299
16.10.11	Plot all Combinations of Hyperparameters	299
17	HPT: sklearn SVC VBDP Data	301
17.1	Step 1: Setup	301
17.2	Step 2: Initialization of the Empty <code>fun_control</code> Dictionary	302
17.3	Step 3: PyTorch Data Loading	303
17.3.1	1. Load Data: Classification VBDP	303
17.3.2	Holdout Train and Test Data	303
17.4	Step 4: Specification of the Preprocessing Model	304
17.5	Step 5: Select Model (<code>algorithm</code>) and <code>core_model_hyper_dict</code>	305
17.6	Step 6: Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	307
17.6.1	Modify hyperparameter of type numeric and integer (boolean)	307
17.6.2	Modify hyperparameter of type factor	307
17.6.3	Optimizers	307
17.6.4	Selection of the Objective: Metric and Loss Functions	308

17.7	Step 7: Selection of the Objective (Loss) Function	308
17.7.1	Metric Function	308
17.7.2	Evaluation on Hold-out Data	309
17.8	Step 8: Calling the SPOT Function	310
17.8.1	Preparing the SPOT Call	310
17.8.2	The Objective Function	311
17.8.3	Run the <code>Spot</code> Optimizer	311
17.9	Step 9: Tensorboard	315
17.10	Step 10: Results	315
17.10.1	Show variable importance	316
17.10.2	Get Default Hyperparameters	317
17.10.3	Get SPOT Results	318
17.10.4	Evaluate SPOT Results	319
17.10.5	Handling Non-deterministic Results	320
17.10.6	Evaluation of the Default Hyperparameters	320
17.10.7	Plot: Compare Predictions	321
17.10.8	Cross-validated Evaluations	322
17.10.9	Detailed Hyperparameter Plots	323
17.10.10	Parallel Coordinates Plot	324
17.10.11	Plot all Combinations of Hyperparameters	324
18	HPT: sklearn KNN Classifier VBDP Data	325
18.1	Step 1: Setup	325
18.2	Step 2: Initialization of the Empty <code>fun_control</code> Dictionary	326
18.2.1	Load Data: Classification VBDP	326
18.2.2	Holdout Train and Test Data	327
18.3	Step 4: Specification of the Preprocessing Model	328
18.4	Step 5: Select Model (<code>algorithm</code>) and <code>core_model_hyper_dict</code>	329
18.5	Step 6: Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	330
18.5.1	Modify hyperparameter of type numeric and integer (boolean)	330
18.5.2	Modify hyperparameter of type factor	331
18.5.3	Optimizers	331
18.5.4	Selection of the Objective: Metric and Loss Functions	331
18.6	Step 7: Selection of the Objective (Loss) Function	331
18.6.1	Metric Function	332
18.6.2	Evaluation on Hold-out Data	333
18.7	Step 8: Calling the SPOT Function	333
18.7.1	Preparing the SPOT Call	333
18.7.2	The Objective Function	334
18.7.3	Run the <code>Spot</code> Optimizer	334
18.8	Step 9: Tensorboard	338

18.9	Step 10: Results	338
18.9.1	Show variable importance	339
18.9.2	Get Default Hyperparameters	340
18.9.3	Get SPOT Results	341
18.9.4	Evaluate SPOT Results	341
18.9.5	Handling Non-deterministic Results	342
18.9.6	Evaluation of the Default Hyperparameters	343
18.9.7	Plot: Compare Predictions	343
18.9.8	Cross-validated Evaluations	345
18.9.9	Detailed Hyperparameter Plots	346
18.9.10	Parallel Coordinates Plot	349
18.9.11	Plot all Combinations of Hyperparameters	349
19	HPT PyTorch: Regression	350
19.1	Step 1: Setup	350
19.2	Step 2: Initialization of the <code>fun_control</code> Dictionary	352
19.3	Step 3: PyTorch Data Loading	352
19.4	Step 4: Specification of the Preprocessing Model	354
19.5	Step 5: Select Model (<code>algorithm</code>) and <code>core_model_hyper_dict</code>	354
19.5.1	Implementing a Configurable Neural Network With <code>spotPython</code>	354
19.5.2	The Search Space	356
19.5.3	Configuring the Search Space With <code>spotPython</code>	356
19.6	Step 6: Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	358
19.6.1	Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	358
19.6.2	Optimizers	358
19.7	Step 7: Selection of the Objective (Loss) Function	359
19.7.1	Evaluation	359
19.7.2	Loss Functions and Metrics	359
19.7.3	Metric	359
19.8	Step 8: Calling the SPOT Function	359
19.8.1	Preparing the SPOT Call	359
19.8.2	The Objective Function <code>fun_torch</code>	360
19.8.3	Starting the Hyperparameter Tuning	360
19.9	Step 9: Tensorboard	464
19.10	Step 10: Results	464
19.10.1	Get the Tuned Architecture (SPOT Results)	466
19.10.2	Evaluation of the Tuned Architecture	467
19.10.3	Cross-validated Evaluations	469
19.10.4	Detailed Hyperparameter Plots	499
19.10.5	Parallel Coordinates Plot	501
19.11	Summary and Outlook	501

20 HPT: PyTorch With VBDP	503
20.1 Step 1: Setup	504
20.2 Step 2: Initialization of the <code>fun_control</code> Dictionary	505
20.3 Step 3: PyTorch Data Loading	505
20.3.1 1. Load VBDP Data	505
20.3.2 Check content of the target column	506
20.4 Step 4: Specification of the Preprocessing Model	507
20.5 Step 5: Select <code>algorithm</code> and <code>core_model_hyper_dict</code>	508
20.5.1 Implementing a Configurable Neural Network With <code>spotPython</code>	508
20.5.2 Add the NN Model to the <code>fun_control</code> Dictionary	508
20.6 Step 6: Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	510
20.6.1 Optimizers	510
20.7 Step 7: Selection of the Objective (Loss) Function	511
20.7.1 Evaluation	511
20.7.2 Loss Functions and Metrics	511
20.7.3 Metric	511
20.8 Step 8: Calling the SPOT Function	512
20.8.1 Preparing the SPOT Call	512
20.8.2 The Objective Function <code>fun_torch</code>	513
20.8.3 Starting the Hyperparameter Tuning	513
20.9 Step 9: Tensorboard	518
20.10 Step 10: Results	519
20.10.1 Get the Tuned Architecture	520
20.10.2 Evaluation of the Tuned Architecture	521
20.10.3 Cross-validated Evaluations	522
20.10.4 Detailed Hyperparameter Plots	525
20.10.5 Parallel Coordinates Plot	525
20.10.6 Plot all Combinations of Hyperparameters	526
 21 HPT PyTorch Lightning: VBDP	 527
21.1 Step 1: Setup	528
21.2 Step 2: Initialization of the <code>fun_control</code> Dictionary	529
21.3 Step 3: PyTorch Data Loading	529
21.3.1 Lightning Dataset and <code>DataModule</code>	529
21.4 Step 4: Specification of the Preprocessing Model	531
21.5 Step 5: Select the NN Model (<code>algorithm</code>) and <code>core_model_hyper_dict</code>	531
21.5.1 Implementing a Configurable Neural Network With <code>spotPython</code>	531
21.5.2 Add the NN Model to the <code>fun_control</code> Dictionary	532
21.6 Step 6: Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	534
21.7 Step 7: Data Splitting, the Objective (Loss) Function and the Metric	536
21.7.1 Evaluation	536

21.7.2	Loss Functions and Metrics	536
21.7.3	Metric	536
21.8	Step 8: Calling the SPOT Function	537
21.8.1	Preparing the SPOT Call	537
21.8.2	The Objective Function <code>fun</code>	538
21.8.3	Starting the Hyperparameter Tuning	541
21.9	Step 9: Tensorboard	563
21.10	Step 10: Results	563
21.10.1	Get the Tuned Architecture	564
21.10.2	Cross Validation With Lightning	566
21.10.3	Detailed Hyperparameter Plots	574
21.10.4	Parallel Coordinates Plot	578
21.10.5	Plot all Combinations of Hyperparameters	579
21.10.6	Visualizing the Activation Distribution	579
22	Submission	582
23	Documentation of the Sequential Parameter Optimization	586
23.1	Example: <code>spot</code>	586
23.1.1	The Objective Function	586
23.1.2	External Parameters	588
23.2	The <code>fun_control</code> Dictionary	591
23.3	The <code>design_control</code> Dictionary	591
23.4	The <code>surrogate_control</code> Dictionary	592
23.5	The <code>optimizer_control</code> Dictionary	592
23.6	Run	593
23.7	Print the Results	595
23.8	Show the Progress	595
23.9	Visualize the Surrogate	595
23.10	Init: Build Initial Design	596
23.11	Replicability	597
23.12	Surrogates	598
23.12.1	A Simple Predictor	598
23.13	Demo/Test: Objective Function Fails	598
23.14	PyTorch: Detailed Description of the Data Splitting	600
23.14.1	Description of the " <code>train_hold_out</code> " Setting	600
References		611

Preface

The goal of hyperparameter tuning (or hyperparameter optimization) is to optimize the hyperparameters to improve the performance of the machine or deep learning model.

spotPython (“Sequential Parameter Optimization Toolbox in Python”) is the Python version of the well-known hyperparameter tuner SPOT, which has been developed in the R programming environment for statistical analysis for over a decade. The related open-access book is available here: [Hyperparameter Tuning for Machine and Deep Learning with R—A Practical Guide](#).

[scikit-learn](#) is a Python module for machine learning built on top of SciPy and is distributed under the 3-Clause BSD license. The project was started in 2007 by David Cournapeau as a Google Summer of Code project, and since then many volunteers have contributed.

[PyTorch](#) is an optimized tensor library for deep learning using GPUs and CPUs.

[River](#) is a Python library for online machine learning. It is designed to be used in real-world environments, where not all data is available at once, but streaming in.

! Important: This book is still under development.

Citation

If this document has been useful to you and you wish to cite it in a scientific publication, please refer to the following paper, which can be found on arXiv: <https://arxiv.org/abs/2305.11930>.

```
@ARTICLE{bart23earxiv,  
  author = {{Bartz-Beielstein}, Thomas},  
  title = "{PyTorch Hyperparameter Tuning -- A Tutorial for spotPython}",  
  journal = {arXiv e-prints},  
  keywords = {Computer Science - Machine Learning, Computer Science - Artificial Intelligence},  
  year = 2023,  
  month = may,  
  eid = {arXiv:2305.11930},
```

```
    pages = {arXiv:2305.11930},
    doi = {10.48550/arXiv.2305.11930},
archivePrefix = {arXiv},
    eprint = {2305.11930},
primaryClass = {cs.LG},
    adsurl = {https://ui.adsabs.harvard.edu/abs/2023arXiv230511930B},
    adsnote = {Provided by the SAO/NASA Astrophysics Data System}
}
```


1 Introduction: Hyperparameter Tuning

Hyperparameter tuning is an important, but often difficult and computationally intensive task. Changing the architecture of a neural network or the learning rate of an optimizer can have a significant impact on the performance.

The goal of hyperparameter tuning is to optimize the hyperparameters in a way that improves the performance of the machine learning or deep learning model. The simplest, but also most computationally expensive, approach uses manual search (or trial-and-error (Meignan et al. 2015)). Commonly encountered is simple random search, i.e., random and repeated selection of hyperparameters for evaluation, and lattice search (“grid search”). In addition, methods that perform directed search and other model-free algorithms, i.e., algorithms that do not explicitly rely on a model, e.g., evolution strategies (Bartz-Beielstein et al. 2014) or pattern search (Lewis, Torczon, and Trosset 2000) play an important role. Also, “hyperband”, i.e., a multi-armed bandit strategy that dynamically allocates resources to a set of random configurations and uses successive bisections to stop configurations with poor performance (Li et al. 2016), is very common in hyperparameter tuning. The most sophisticated and efficient approaches are the Bayesian optimization and surrogate model based optimization methods, which are based on the optimization of cost functions determined by simulations or experiments.

We consider below a surrogate model based optimization-based hyperparameter tuning approach based on the Python version of the SPOT (“Sequential Parameter Optimization Toolbox”) (Bartz-Beielstein, Lasarczyk, and Preuss 2005), which is suitable for situations where only limited resources are available. This may be due to limited availability and cost of hardware, or due to the fact that confidential data may only be processed locally, e.g., due to legal requirements. Furthermore, in our approach, the understanding of algorithms is seen as a key tool for enabling transparency and explainability. This can be enabled, for example, by quantifying the contribution of machine learning and deep learning components (nodes, layers, split decisions, activation functions, etc.). Understanding the importance of hyperparameters and the interactions between multiple hyperparameters plays a major role in the interpretability and explainability of machine learning models. SPOT provides statistical tools for understanding hyperparameters and their interactions. Last but not least, it should be noted that the SPOT software code is available in the open source `spotPython` package on github¹, allowing replicability of the results. This tutorial describes the Python variant of SPOT, which is called

¹<https://github.com/sequential-parameter-optimization>

`spotPython`. The R implementation is described in Bartz et al. (2022). SPOT is an established open source software that has been maintained for more than 15 years (Bartz-Beielstein, Lasarczyk, and Preuss 2005) (Bartz et al. 2022).

This tutorial is structured as follows. The concept of the hyperparameter tuning software `spotPython` is described in Section 1.1. Chapter 14 describes the execution of the example from the tutorial “Hyperparameter Tuning with Ray Tune” (PyTorch 2023a). The integration of `spotPython` into the PyTorch training workflow is described in detail in the following sections. Section 14.1 describes the setup of the tuners. Section 14.3 describes the data loading. Section 14.5 describes the model to be tuned. The search space is introduced in Section 14.5.3. Optimizers are presented in Section 14.6.1. How to split the data in train, validation, and test sets is described in Section 14.7.1. The selection of the loss function and metrics is described in Section 14.7.5. Section 14.8.1 describes the preparation of the `spotPython` call. The objective function is described in Section 14.8.2. How to use results from previous runs and default hyperparameter configurations is described in Section 14.8.3. Starting the tuner is shown in Section 14.8.4. TensorBoard can be used to visualize the results as shown in Section 14.9. Results are discussed and explained in Section 14.10.

?@sec-hyperparameter-tuning-lightning-30 shows the integration of `spotPython` into the PyTorch Lightning training workflow.

Section 14.11 presents a summary and an outlook.

i Note

The corresponding `.ipynb` notebook (Bartz-Beielstein 2023) is updated regularly and reflects updates and changes in the `spotPython` package. It can be downloaded from https://github.com/sequential-parameter-optimization/spotPython/blob/main/notebooks/14_spot_ray_hpt_torch_cifar10.ipynb.

1.1 The Hyperparameter Tuning Software SPOT

Surrogate model based optimization methods are common approaches in simulation and optimization. SPOT was developed because there is a great need for sound statistical analysis of simulation and optimization algorithms. SPOT includes methods for tuning based on classical regression and analysis of variance techniques. It presents tree-based models such as classification and regression trees and random forests as well as Bayesian optimization (Gaussian process models, also known as Kriging). Combinations of different meta-modeling approaches are possible. SPOT comes with a sophisticated surrogate model based optimization method, that can handle discrete and continuous inputs. Furthermore, any model implemented in `scikit-learn` can be used out-of-the-box as a surrogate in `spotPython`.

SPOT implements key techniques such as exploratory fitness landscape analysis and sensitivity analysis. It can be used to understand the performance of various algorithms, while simultaneously giving insights into their algorithmic behavior. In addition, SPOT can be used as an optimizer and for automatic and interactive tuning. Details on SPOT and its use in practice are given by Bartz et al. (2022).

A typical hyperparameter tuning process with `spotPython` consists of the following steps:

1. Loading the data (training and test datasets), see Section 14.3.
2. Specification of the preprocessing model, see Section 14.4. This model is called `prep_model` (“preparation” or pre-processing). The information required for the hyperparameter tuning is stored in the dictionary `fun_control`. Thus, the information needed for the execution of the hyperparameter tuning is available in a readable form.
3. Selection of the machine learning or deep learning model to be tuned, see Section 14.5. This is called the `core_model`. Once the `core_model` is defined, then the associated hyperparameters are stored in the `fun_control` dictionary. First, the hyperparameters of the `core_model` are initialized with the default values of the `core_model`. As default values we use the default values contained in the `spotPython` package for the algorithms of the `torch` package.
4. Modification of the default values for the hyperparameters used in `core_model`, see Section 14.6.0.1. This step is optional.
 1. numeric parameters are modified by changing the bounds.
 2. categorical parameters are modified by changing the categories (“levels”).
5. Selection of target function (loss function) for the optimizer, see Section 14.7.5.
6. Calling SPOT with the corresponding parameters, see Section 14.8.4. The results are stored in a dictionary and are available for further analysis.
7. Presentation, visualization and interpretation of the results, see Section 14.10.

1.2 Spot as an Optimizer

The `spot` loop consists of the following steps:

1. Init: Build initial design X
2. Evaluate initial design on real objective f : $y = f(X)$
3. Build surrogate: $S = S(X, y)$
4. Optimize on surrogate: $X_0 = \text{optimize}(S)$
5. Evaluate on real objective: $y_0 = f(X_0)$
6. Impute (Infill) new points: $X = X \cup X_0$, $y = y \cup y_0$.
7. Got 3.

Central Idea: Evaluation of the surrogate model S is much cheaper (or / and much faster) than running the real-world experiment f . We start with a small example.

1.3 Example: Spot and the Sphere Function

```
import numpy as np
from math import inf
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from scipy.optimize import shgo
from scipy.optimize import direct
from scipy.optimize import differential_evolution
import matplotlib.pyplot as plt
```

1.3.1 The Objective Function: Sphere

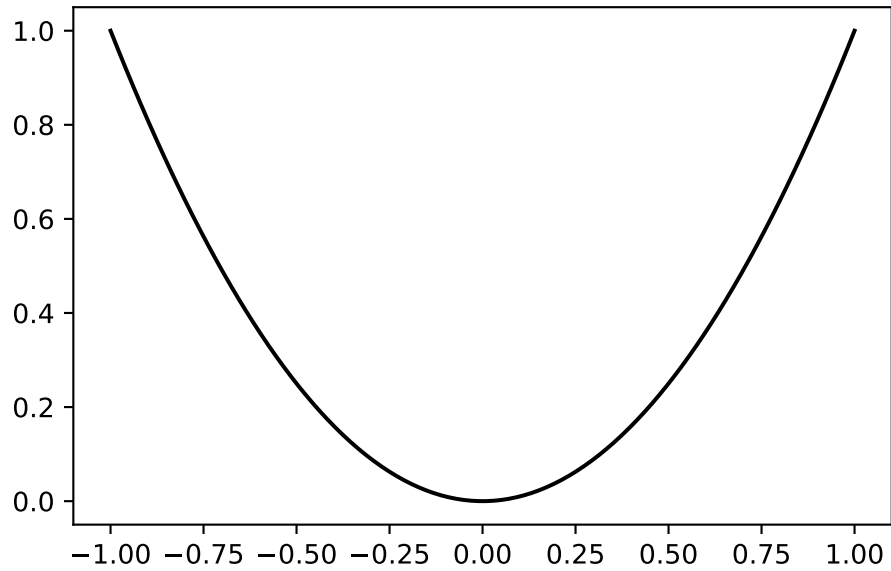
The `spotPython` package provides several classes of objective functions. We will use an analytical objective function, i.e., a function that can be described by a (closed) formula:

$$f(x) = x^2$$

```
fun = analytical().fun_sphere
```

We can apply the function `fun` to input values and plot the result:

```
x = np.linspace(-1,1,100).reshape(-1,1)
y = fun(x)
plt.figure()
plt.plot(x, y, "k")
plt.show()
```



```
spot_0 = spot.Spot(fun=fun,  
                  lower = np.array([-1]),  
                  upper = np.array([1]))
```

```
spot_0.run()
```

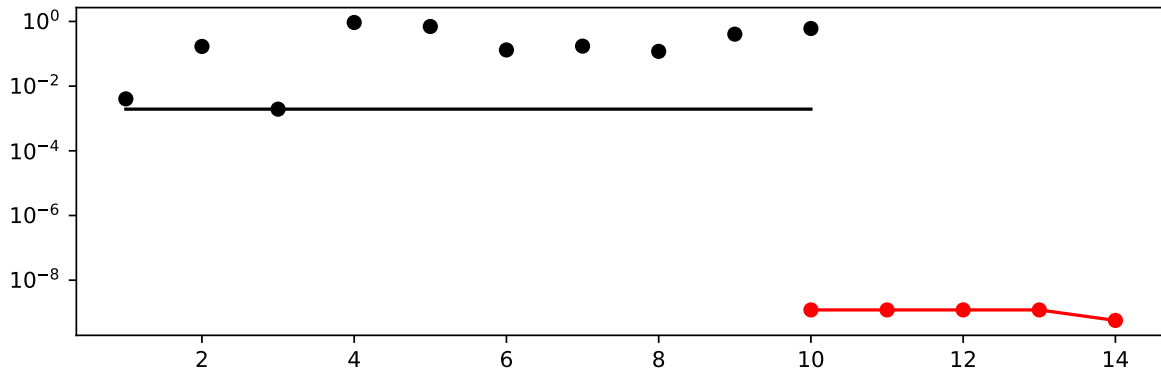
```
<spotPython.spot.spot.Spot at 0x153eea590>
```

```
spot_0.print_results()
```

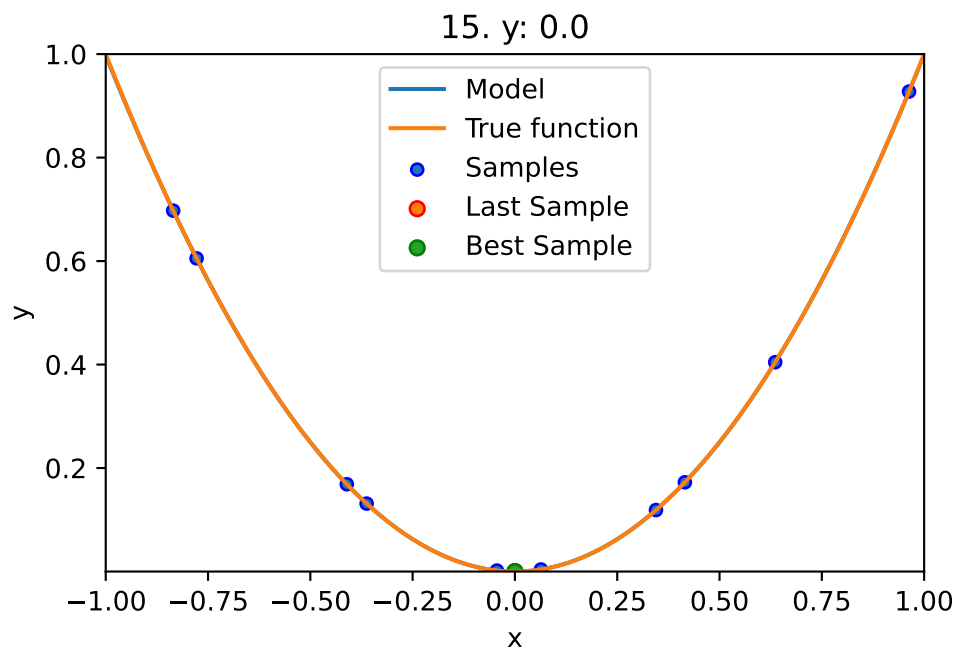
```
min y: 5.69019918867849e-10  
x0: 2.3854138401288967e-05
```

```
[['x0', 2.3854138401288967e-05]]
```

```
spot_0.plot_progress(log_y=True)
```



```
spot_0.plot_model()
```



1.4 Spot Parameters: fun_evals, init_size and show_models

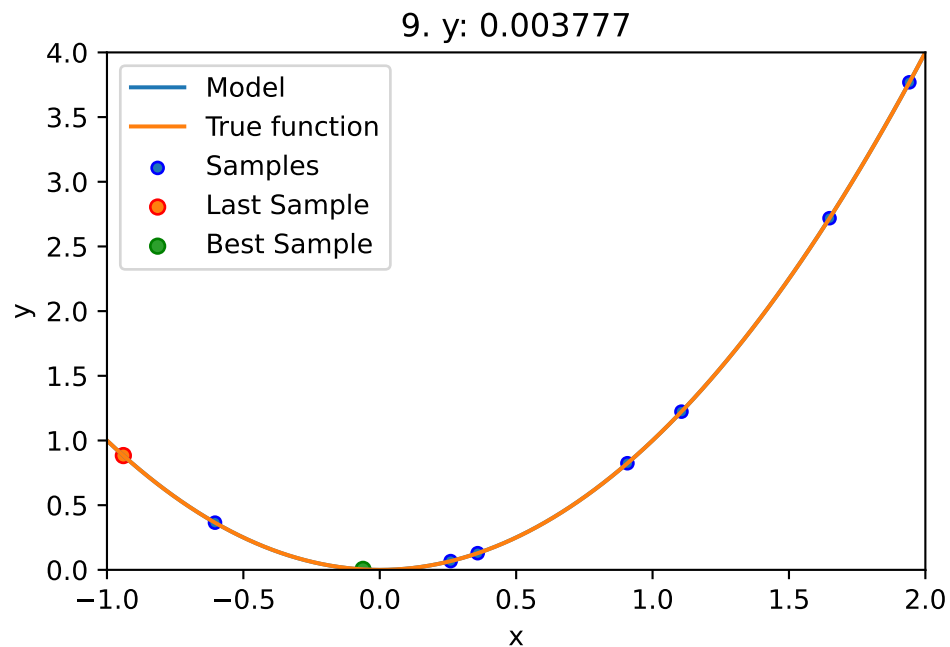
We will modify three parameters:

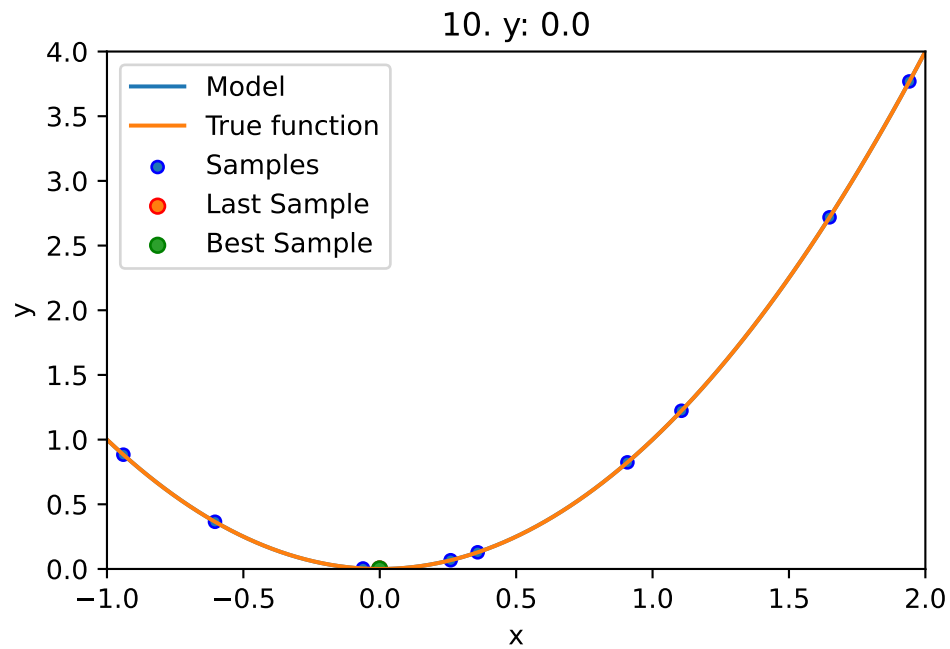
1. The number of function evaluations (`fun_evals`)
2. The size of the initial design (`init_size`)

3. The parameter `show_models`, which visualizes the search process for 1-dim functions.

The full list of the `Spot` parameters is shown in the Help System and in the notebook `spot_doc.ipynb`.

```
spot_1 = spot.Spot(fun=fun,  
                  lower = np.array([-1]),  
                  upper = np.array([2]),  
                  fun_evals= 10,  
                  seed=123,  
                  show_models=True,  
                  design_control={"init_size": 9})  
  
spot_1.run()
```





<spotPython.spot.spot.Spot at 0x15891b3d0>

1.5 Print the Results

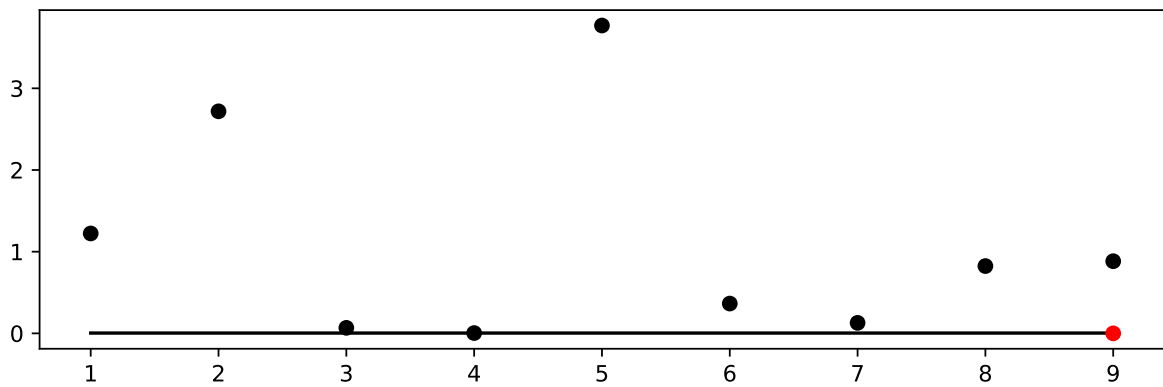
```
spot_1.print_results()
```

```
min y: 3.6858846844978905e-07  
x0: -0.0006071148725321997
```

```
[['x0', -0.0006071148725321997]]
```

1.6 Show the Progress

```
spot_1.plot_progress()
```

2 Multi-dimensional Functions

This notebook illustrates how high-dimensional functions can be analyzed.

2.1 Example: Spot and the 3-dim Sphere Function

```
import numpy as np
from math import inf
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from scipy.optimize import shgo
from scipy.optimize import direct
from scipy.optimize import differential_evolution
import matplotlib.pyplot as plt
import pylab
from numpy import append, ndarray, multiply, isinf, linspace, meshgrid, ravel
from numpy import array
```

2.1.1 The Objective Function: 3-dim Sphere

- The spotPython package provides several classes of objective functions.
- We will use an analytical objective function, i.e., a function that can be described by a (closed) formula:

$$f(x) = \sum_i^n x_i^2$$

- Here we will use $n = 3$.

```
fun = analytical().fun_sphere
```

- The size of the lower bound vector determines the problem dimension.
- Here we will use `np.array([-1, -1, -1])`, i.e., a three-dim function.

- We will use three different `theta` values (one for each dimension), i.e., we set `surrogate_control={"n_theta": 3}`.

```
spot_3 = spot.Spot(fun=fun,
                  lower = -1.0*np.ones(3),
                  upper = np.ones(3),
                  var_name=["Pressure", "Temp", "Lambda"],
                  show_progress=True,
                  surrogate_control={"n_theta": 3})

spot_3.run()
```

```
spotPython tuning: 0.03443399805488846 [#####---] 73.33%
```

```
spotPython tuning: 0.03134895672225177 [#####--] 80.00%
```

```
spotPython tuning: 0.0009630555620661592 [#####-] 86.67%
```

```
spotPython tuning: 8.567364874637509e-05 [#####-] 93.33%
```

```
spotPython tuning: 6.0300780324366926e-05 [#####] 100.00% Done...
```

```
<spotPython.spot.spot.Spot at 0x108d8f550>
```

2.1.2 Results

```
spot_3.print_results()
```

```
min y: 6.0300780324366926e-05
```

```
Pressure: 0.00514742089151478
```

```
Temp: 0.001954003740617489
```

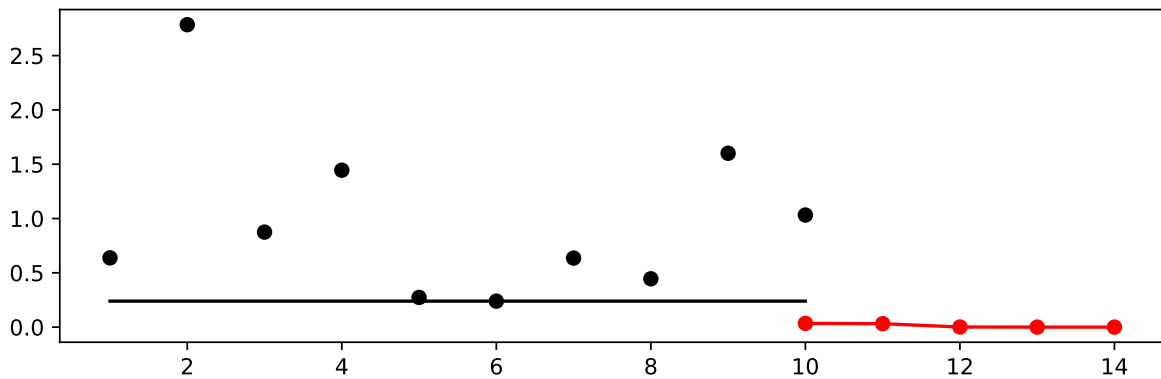
```
Lambda: 0.005476012040857559
```

```
[['Pressure', 0.00514742089151478],
```

```
 ['Temp', 0.001954003740617489],
```

```
 ['Lambda', 0.005476012040857559]]
```

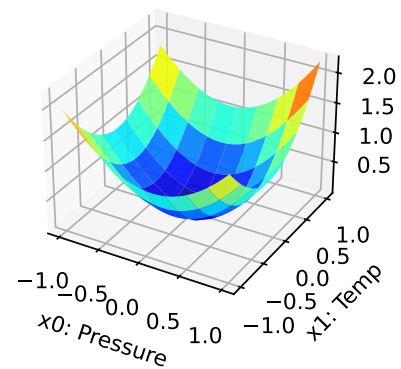
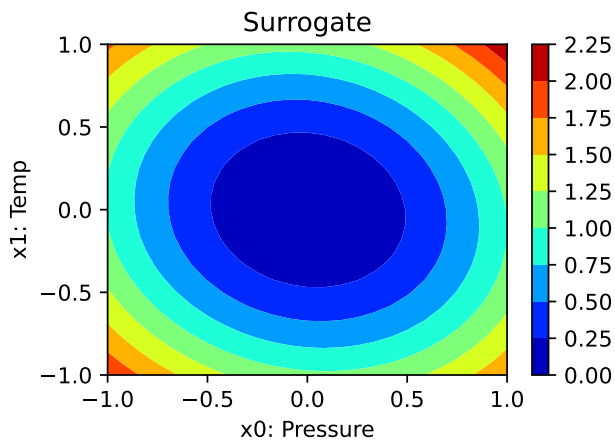
```
spot_3.plot_progress()
```



2.1.3 A Contour Plot

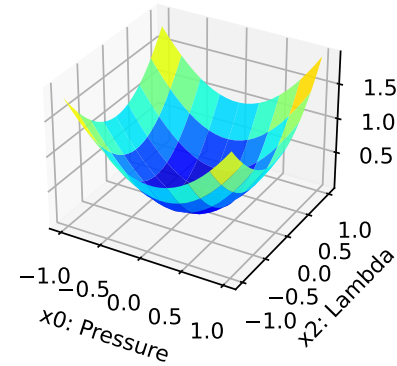
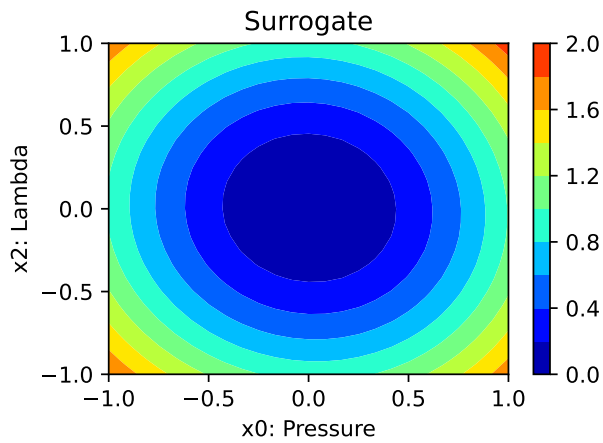
- We can select two dimensions, say $i = 0$ and $j = 1$, and generate a contour plot as follows.
 - Note: We have specified identical `min_z` and `max_z` values to generate comparable plots!

```
spot_3.plot_contour(i=0, j=1, min_z=0, max_z=2.25)
```



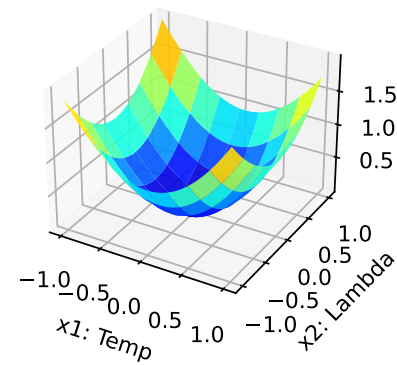
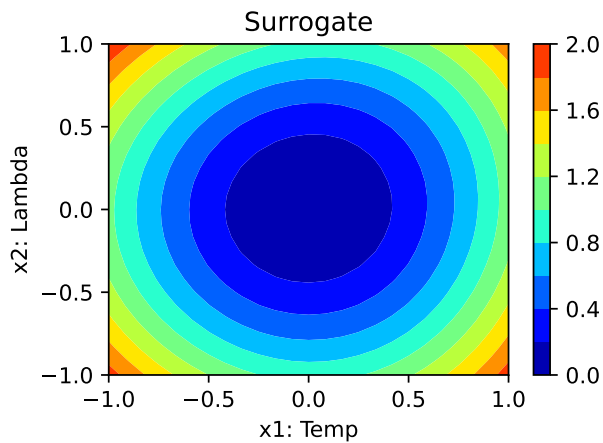
- In a similar manner, we can plot dimension $i = 0$ and $j = 2$:

```
spot_3.plot_contour(i=0, j=2, min_z=0, max_z=2.25)
```



- The final combination is $i = 1$ and $j = 2$:

```
spot_3.plot_contour(i=1, j=2, min_z=0, max_z=2.25)
```



- The three plots look very similar, because the `fun_sphere` is symmetric.
- This can also be seen from the variable importance:

```
spot_3.print_importance()
```

```
Pressure: 100.0
Temp: 99.69922253450551
```

Lambda: 93.68147774373058

```
[['Pressure', 100.0],  
 ['Temp', 99.69922253450551],  
 ['Lambda', 93.68147774373058]]
```

2.2 Conclusion

Based on this quick analysis, we can conclude that all three dimensions are equally important (as expected, because the analytical function is known).

2.3 Exercises

- Important:
 - Results from these exercises should be added to this document, i.e., you should submit an updated version of this notebook.
 - Please combine your results using this notebook.
 - Only one notebook from each group!
 - Presentation is based on this notebook. No additional slides are required!
 - spotPython version 0.16.11 (or greater) is required

2.3.1 The Three Dimensional fun_cubed

- The input dimension is 3. The search range is $-1 \leq x \leq 1$ for all dimensions.
- Generate contour plots
- Calculate the variable importance.
- Discuss the variable importance:
 - Are all variables equally important?
 - If not:
 - * Which is the most important variable?
 - * Which is the least important variable?

2.3.2 The Ten Dimensional `fun_wing_wt`

- The input dimension is 10. The search range is $0 \leq x \leq 1$ for all dimensions.
- Calculate the variable importance.
- Discuss the variable importance:
 - Are all variables equally important?
 - If not:
 - * Which is the most important variable?
 - * Which is the least important variable?
 - Generate contour plots for the three most important variables. Do they confirm your selection?

2.3.3 The Three Dimensional `fun_runge`

- The input dimension is 3. The search range is $-5 \leq x \leq 5$ for all dimensions.
- Generate contour plots
- Calculate the variable importance.
- Discuss the variable importance:
 - Are all variables equally important?
 - If not:
 - * Which is the most important variable?
 - * Which is the least important variable?

2.3.4 The Three Dimensional `fun_linear`

- The input dimension is 3. The search range is $-5 \leq x \leq 5$ for all dimensions.
- Generate contour plots
- Calculate the variable importance.
- Discuss the variable importance:
 - Are all variables equally important?
 - If not:
 - * Which is the most important variable?
 - * Which is the least important variable?

3 Isotropic and Anisotropic Kriging

3.1 Example: Isotropic Spot Surrogate and the 2-dim Sphere Function

```
import numpy as np
from math import inf
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from scipy.optimize import shgo
from scipy.optimize import direct
from scipy.optimize import differential_evolution
import matplotlib.pyplot as plt
```

3.1.1 The Objective Function: 2-dim Sphere

- The `spotPython` package provides several classes of objective functions.
- We will use an analytical objective function, i.e., a function that can be described by a (closed) formula:

$$f(x, y) = x^2 + y^2$$

```
fun = analytical().fun_sphere
fun_control = {"sigma": 0,
               "seed": 123}
```

- The size of the `lower` bound vector determines the problem dimension.
- Here we will use `np.array([-1, -1])`, i.e., a two-dim function.

```
spot_2 = spot.Spot(fun=fun,
                   lower = np.array([-1, -1]),
                   upper = np.array([1, 1]))

spot_2.run()
```



```
<spotPython.spot.spot.Spot at 0x1510fe8f0>
```

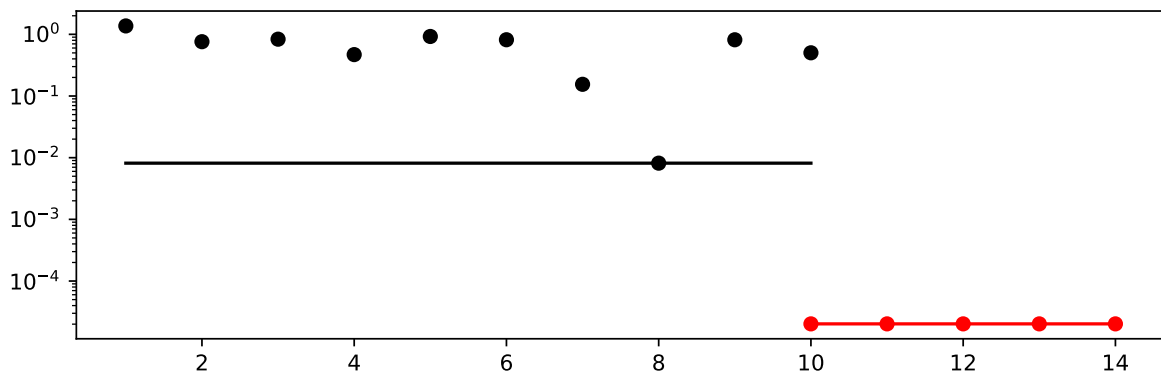
3.1.2 Results

```
spot_2.print_results()
```

```
min y: 2.020789135198605e-05  
x0: 0.0015751963468338146  
x1: 0.004210302580683181
```

```
[['x0', 0.0015751963468338146], ['x1', 0.004210302580683181]]
```

```
spot_2.plot_progress(log_y=True)
```



3.2 Example With Anisotropic Kriging

- The default parameter setting of `spotPython`'s Kriging surrogate uses the same `theta` value for every dimension.
- This is referred to as “using an isotropic kernel”.
- If different `theta` values are used for each dimension, then an anisotropic kernel is used
- To enable anisotropic models in `spotPython`, the number of `theta` values should be larger than one.
- We can use `surrogate_control={"n_theta": 2}` to enable this behavior (2 is the problem dimension).

```
spot_2_anisotropic = spot.Spot(fun=fun,
                                lower = np.array([-1, -1]),
                                upper = np.array([1, 1]),
                                surrogate_control={"n_theta": 2})
spot_2_anisotropic.run()
```

```
<spotPython.spot.spot.Spot at 0x155ca7310>
```

3.2.1 Taking a Look at the `theta` Values

- We can check, whether one or several `theta` values were used.
- The `theta` values from the surrogate can be printed as follows:

```
spot_2_anisotropic.surrogate.theta
```

```
array([0.24805857, 0.35713614])
```

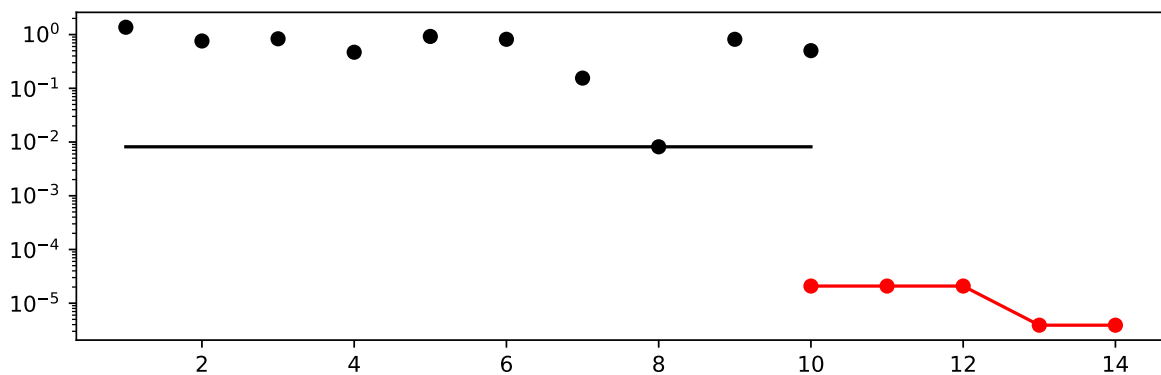
- Since the surrogate from the isotropic setting was stored as `spot_2`, we can also take a look at the `theta` value from this model:

```
spot_2.surrogate.theta
```

```
array([0.26287446])
```

- Next, the search progress of the optimization with the anisotropic model can be visualized:

```
spot_2_anisotropic.plot_progress(log_y=True)
```



```
spot_2_anisotropic.print_results()
```

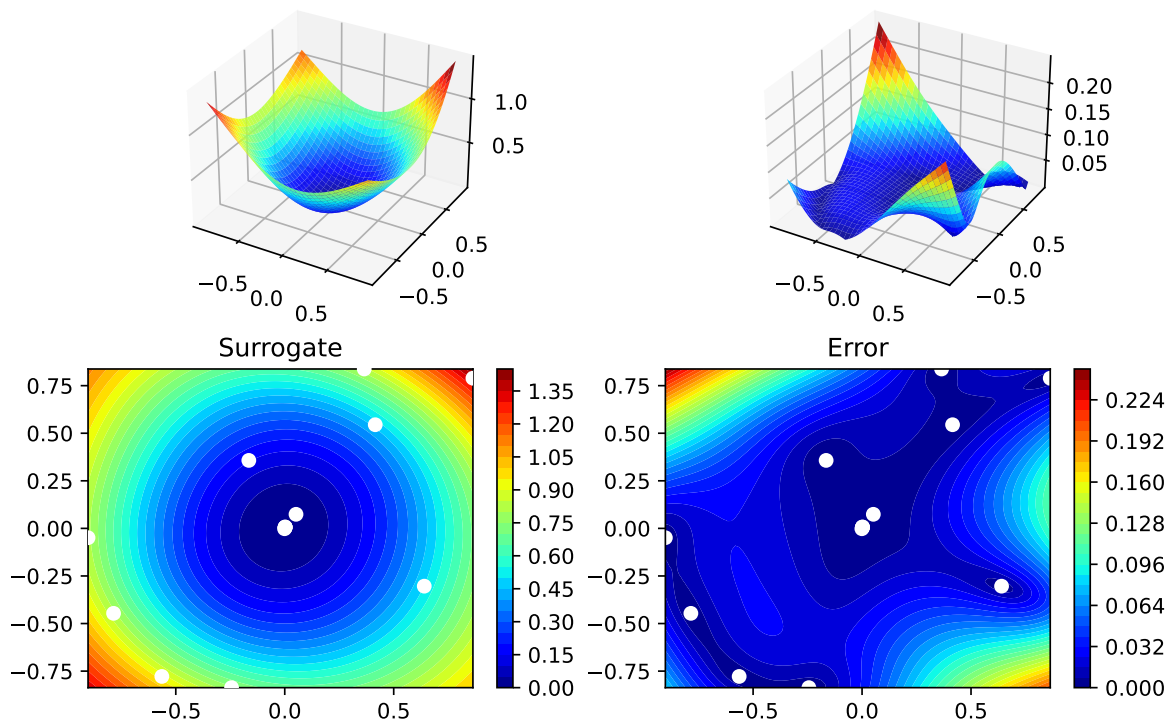
min y: 3.898914658670152e-06

x0: -0.0008031859004420657

x1: -0.0018038312193775835

```
[['x0', -0.0008031859004420657], ['x1', -0.0018038312193775835]]
```

```
spot_2_anisotropic.surrogate.plot()
```



3.3 Exercises

3.3.1 fun_branin

- Describe the function.
 - The input dimension is 2. The search range is $-5 \leq x_1 \leq 10$ and $0 \leq x_2 \leq 15$.

- Compare the results from `spotPython` run a) with isotropic and b) anisotropic surrogate models.
- Modify the termination criterion: instead of the number of evaluations (which is specified via `fun_evals`), the time should be used as the termination criterion. This can be done as follows (`max_time=1` specifies a run time of one minute):

```
fun_evals=inf,
max_time=1,
```

3.3.2 `fun_sin_cos`

- Describe the function.
 - The input dimension is 2. The search range is $-2\pi \leq x_1 \leq 2\pi$ and $-2\pi \leq x_2 \leq 2\pi$.
- Compare the results from `spotPython` run a) with isotropic and b) anisotropic surrogate models.
- Modify the termination criterion (`max_time` instead of `fun_evals`) as described for `fun_branin`.

3.3.3 `fun_runge`

- Describe the function.
 - The input dimension is 2. The search range is $-5 \leq x_1 \leq 5$ and $-5 \leq x_2 \leq 5$.
- Compare the results from `spotPython` run a) with isotropic and b) anisotropic surrogate models.
- Modify the termination criterion (`max_time` instead of `fun_evals`) as described for `fun_branin`.

3.3.4 `fun_wingwt`

- Describe the function.
 - The input dimension is 10. The search ranges are between 0 and 1 (values are mapped internally to their natural bounds).
- Compare the results from `spotPython` run a) with isotropic and b) anisotropic surrogate models.
- Modify the termination criterion (`max_time` instead of `fun_evals`) as described for `fun_branin`.

4 Using sklearn Surrogates in spotPython

This notebook explains how different surrogate models from `scikit-learn` can be used as surrogates in `spotPython` optimization runs.

```
import numpy as np
from math import inf
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from scipy.optimize import shgo
from scipy.optimize import direct
from scipy.optimize import differential_evolution
import matplotlib.pyplot as plt
```

4.1 Example: Branin Function with spotPython's Internal Kriging Surrogate

4.1.1 The Objective Function Branin

- The `spotPython` package provides several classes of objective functions.
- We will use an analytical objective function, i.e., a function that can be described by a (closed) formula.
- Here we will use the Branin function:

$y = a * (x_2 - b * x_1^2 + c * x_1 - r) ** 2 + s * (1 - t) * \cos(x_1) + s$,
where values of a , b , c , r , s and t are: $a = 1$, $b = 5.1 / (4 * \pi^2)$,
 $c = 5 / \pi$, $r = 6$, $s = 10$ and $t = 1 / (8 * \pi)$.

- It has three global minima:

$f(x) = 0.397887$ at $(-\pi, 12.275)$, $(\pi, 2.275)$, and $(9.42478, 2.475)$.

```
from spotPython.fun.objectivefunctions import analytical
lower = np.array([-5,-0])
```

```
upper = np.array([10,15])

fun = analytical().fun_branin
```

4.1.2 Running the surrogate model based optimizer Spot:

```
spot_2 = spot.Spot(fun=fun,
                  lower = lower,
                  upper = upper,
                  fun_evals = 20,
                  max_time = inf,
                  seed=123,
                  design_control={"init_size": 10})
```

```
spot_2.run()
```

```
<spotPython.spot.spot.Spot at 0x1582f2680>
```

4.1.3 Print the Results

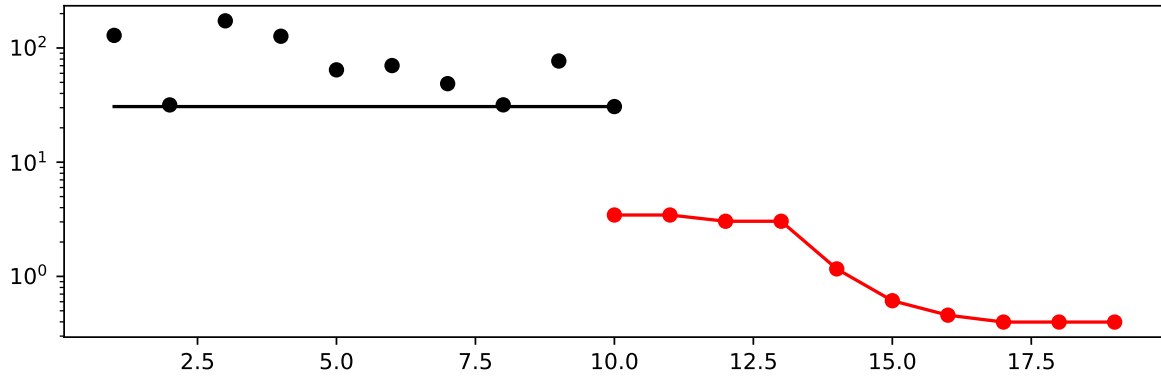
```
spot_2.print_results()
```

```
min y: 0.3982296851228586
x0: 3.135563584477711
x1: 2.2926607128616965
```

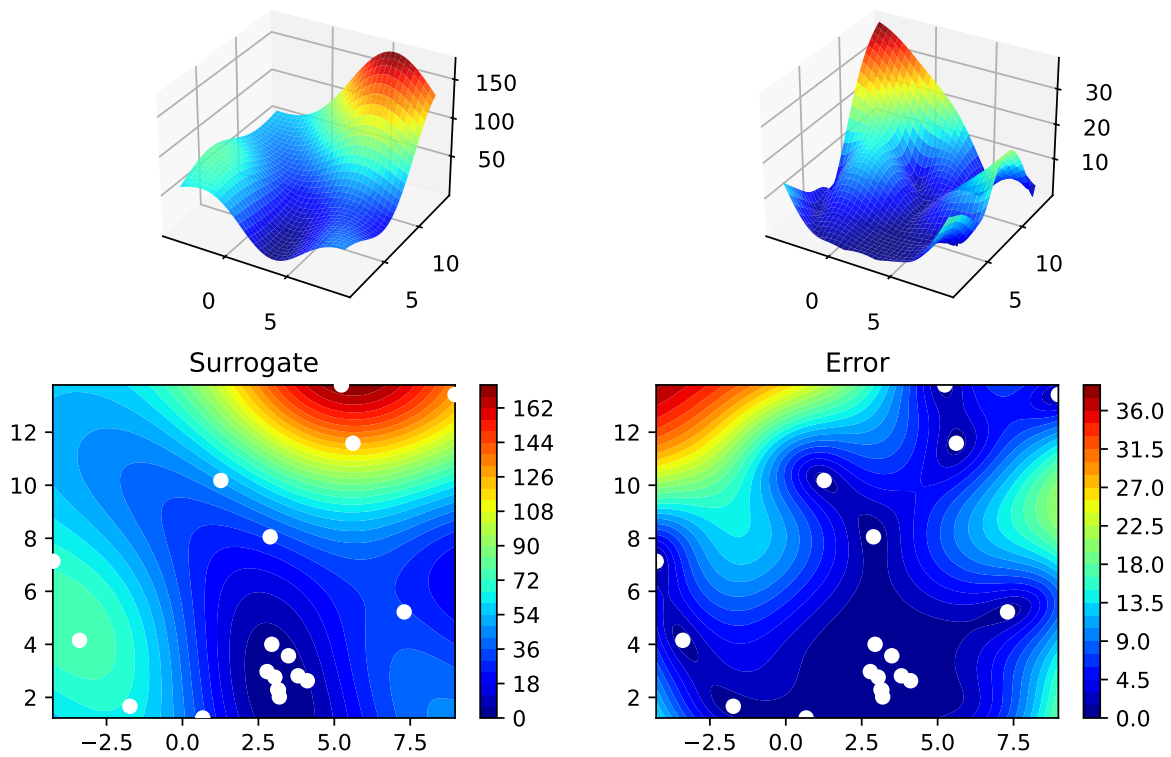
```
[['x0', 3.135563584477711], ['x1', 2.2926607128616965]]
```

4.1.4 Show the Progress and the Surrogate

```
spot_2.plot_progress(log_y=True)
```



```
spot_2.surrogate.plot()
```



4.2 Example: Using Surrogates From scikit-learn

- Default is the `spotPython` (i.e., the internal) `kriging` surrogate.

- It can be called explicitly and passed to `Spot`.

```
from spotPython.build.kriging import Kriging
S_0 = Kriging(name='kriging', seed=123)
```

- Alternatively, models from `scikit-learn` can be selected, e.g., Gaussian Process, RBFs, Regression Trees, etc.

```
# Needed for the sklearn surrogates:
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn import linear_model
from sklearn import tree
import pandas as pd
```

- Here are some additional models that might be useful later:

```
S_Tree = DecisionTreeRegressor(random_state=0)
S_LM = linear_model.LinearRegression()
S_Ridge = linear_model.Ridge()
S_RF = RandomForestRegressor(max_depth=2, random_state=0)
```

4.2.1 GaussianProcessRegressor as a Surrogate

- To use a Gaussian Process model from `sklearn`, that is similar to `spotPython`'s `Kriging`, we can proceed as follows:

```
kernel = 1 * RBF(length_scale=1.0, length_scale_bounds=(1e-2, 1e2))
S_GP = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=9)
```

- The `scikit-learn` GP model `S_GP` is selected for `Spot` as follows:

```
surrogate = S_GP
```

- We can check the kind of surrogate model with the command `isinstance`:

```
isinstance(S_GP, GaussianProcessRegressor)
```

True


```
isinstance(S_0, Kriging)
```

True

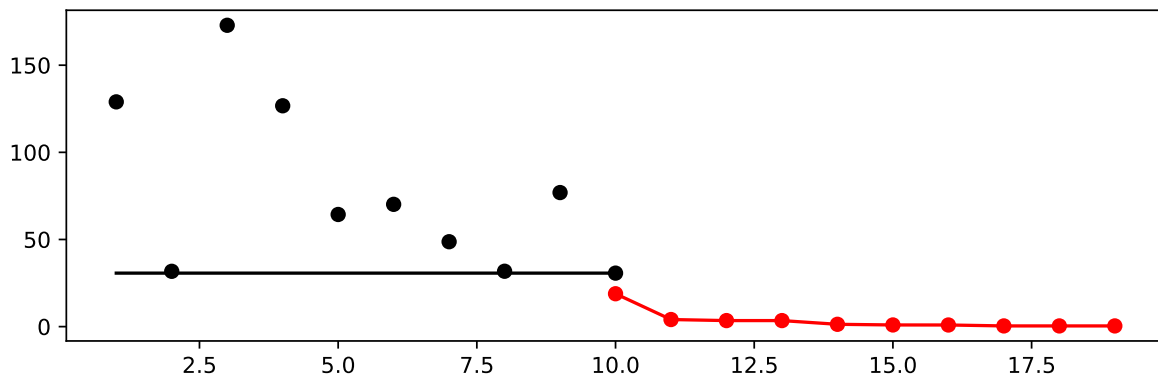
- Similar to the Spot run with the internal Kriging model, we can call the run with the scikit-learn surrogate:

```
fun = analytical(seed=123).fun_branin
spot_2_GP = spot.Spot(fun=fun,
                      lower = lower,
                      upper = upper,
                      fun_evals = 20,
                      seed=123,
                      design_control={"init_size": 10},
                      surrogate = S_GP)

spot_2_GP.run()
```

<spotPython.spot.spot.Spot at 0x15d1d8910>

```
spot_2_GP.plot_progress()
```



```
spot_2_GP.print_results()
```

min y: 0.398269346702568

x0: 3.149609857302586

x1: 2.2773233969124718

```
[['x0', 3.149609857302586], ['x1', 2.2773233969124718]]
```

4.3 Example: One-dimensional Sphere Function With spotPython's Kriging

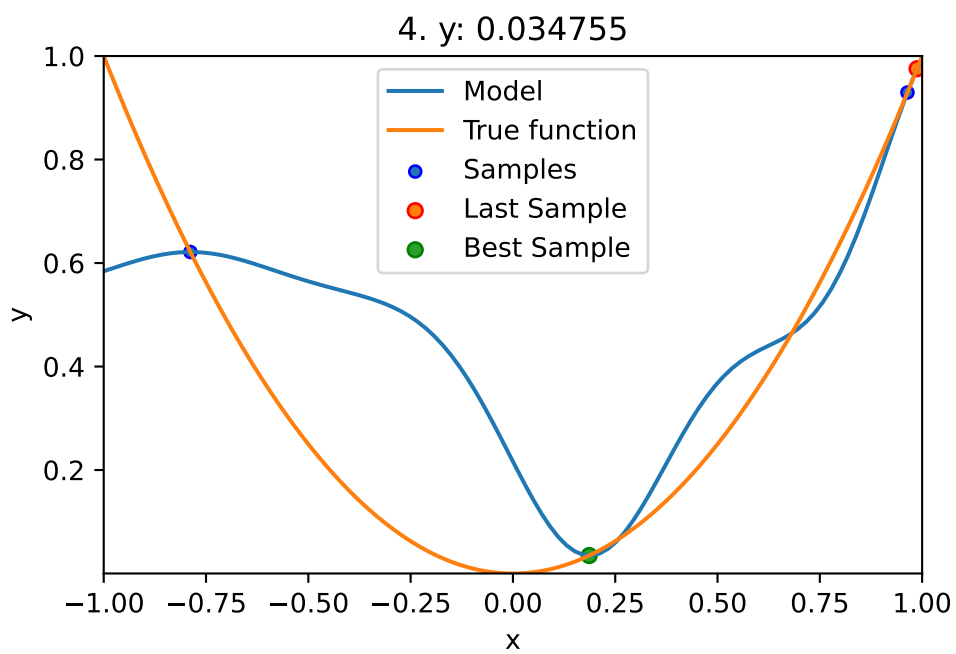
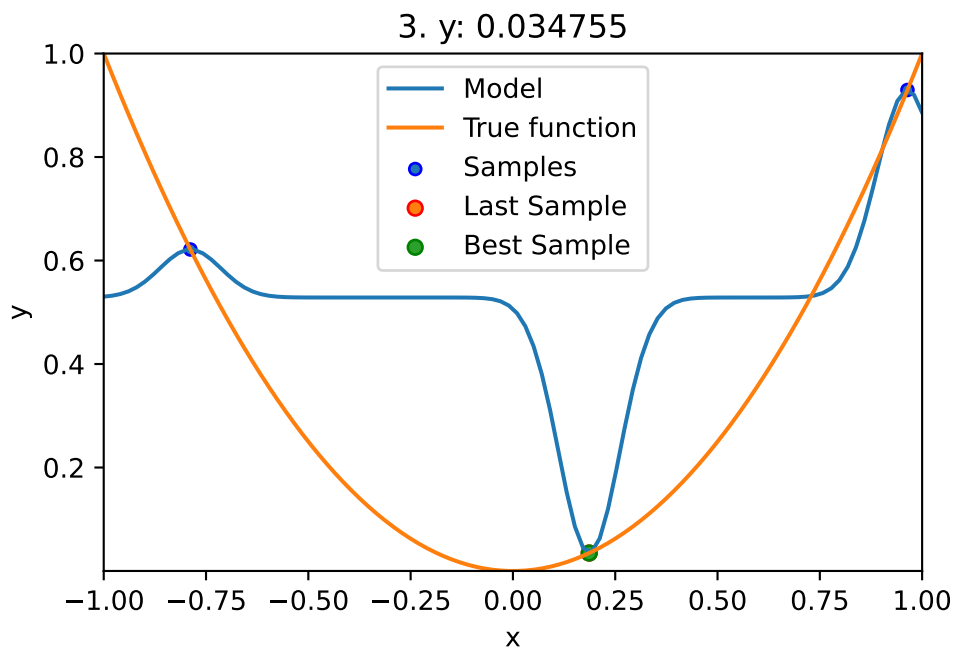
- In this example, we will use an one-dimensional function, which allows us to visualize the optimization process.

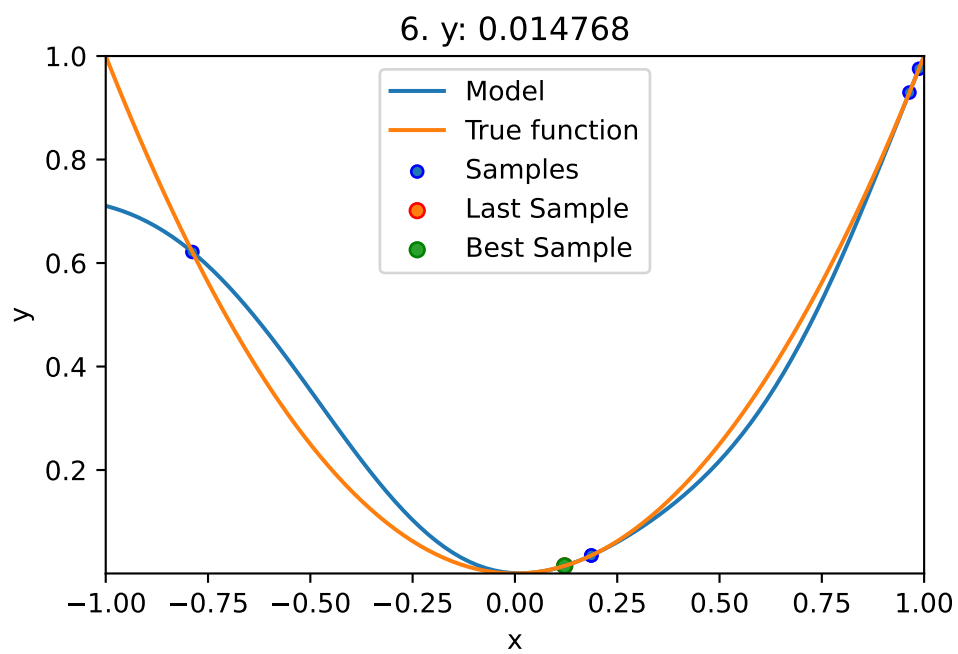
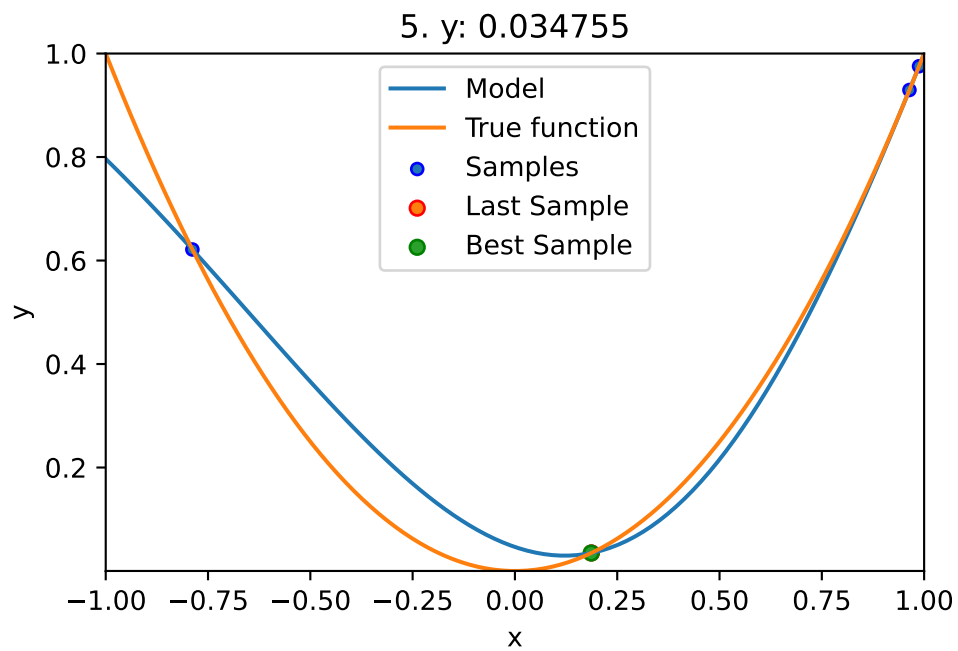
– `show_models= True` is added to the argument list.

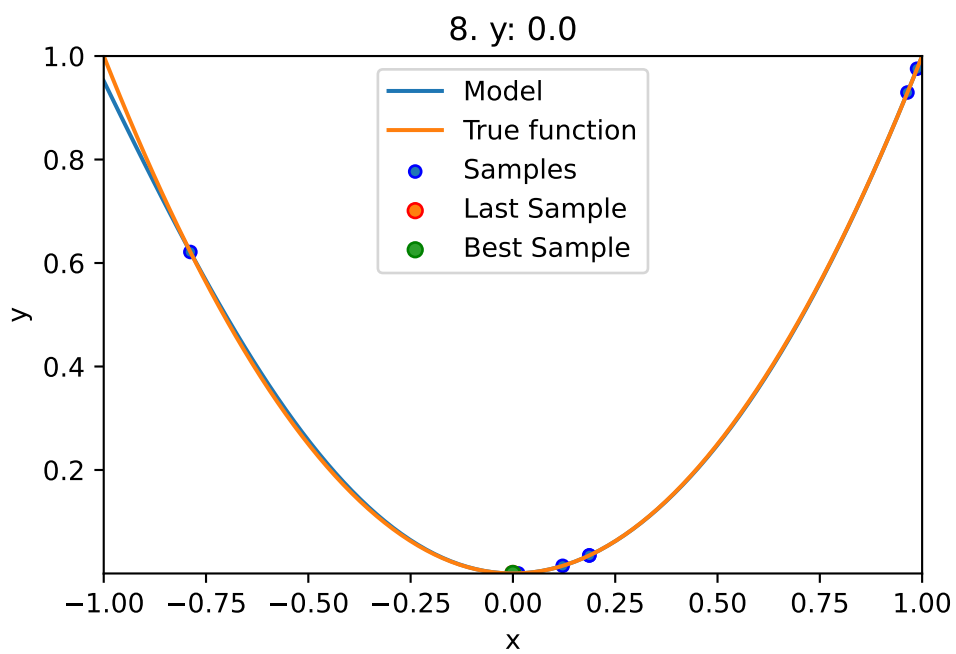
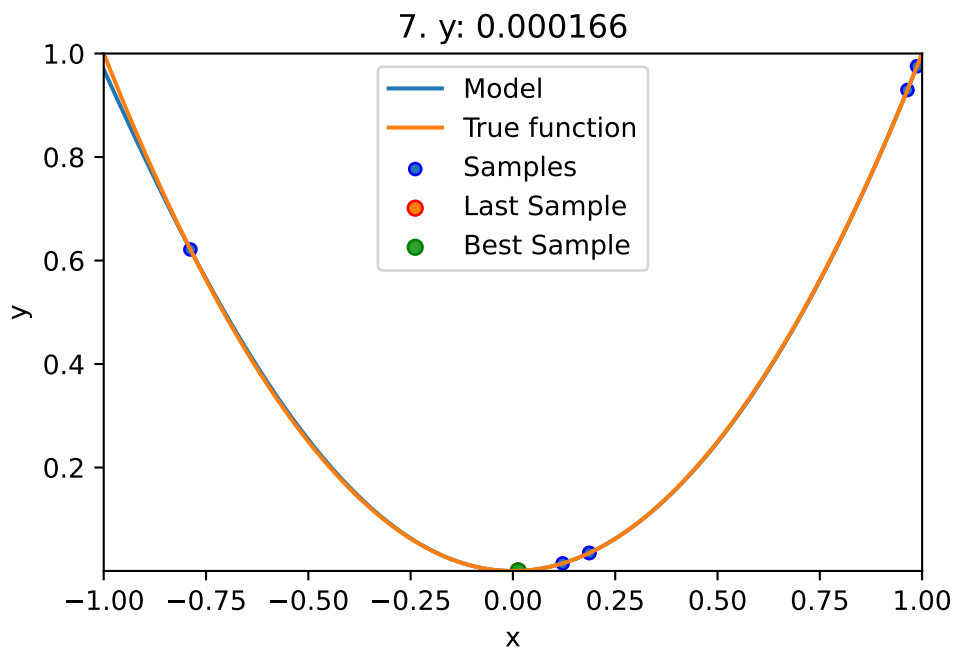
```
from spotPython.fun.objectivefunctions import analytical
lower = np.array([-1])
upper = np.array([1])
fun = analytical(seed=123).fun_sphere
```

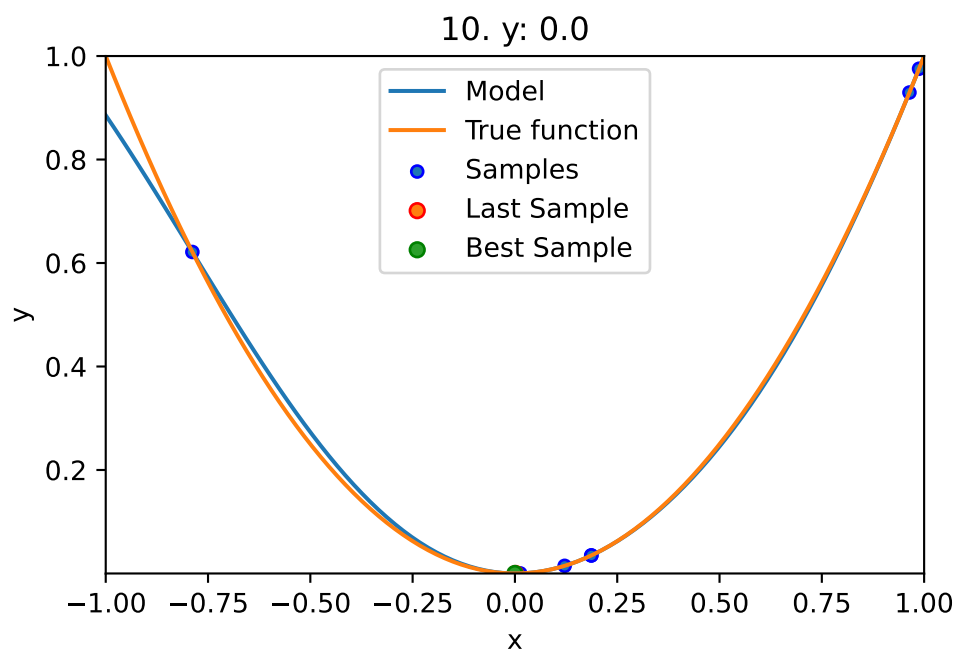
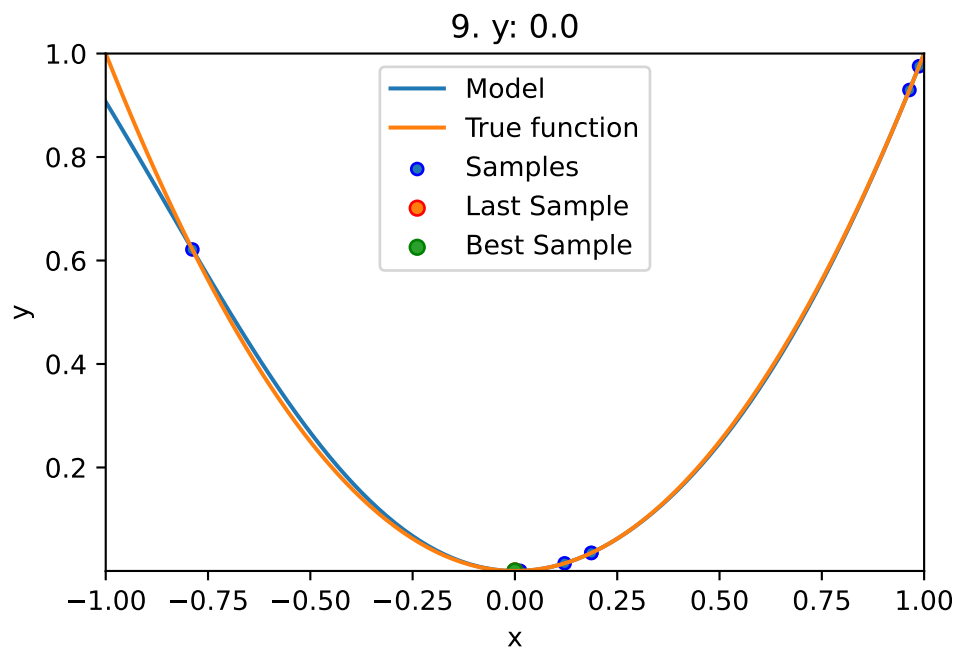
```
spot_1 = spot.Spot(fun=fun,
                   lower = lower,
                   upper = upper,
                   fun_evals = 10,
                   max_time = inf,
                   seed=123,
                   show_models= True,
                   tolerance_x = np.sqrt(np.spacing(1)),
                   design_control={"init_size": 3},)

spot_1.run()
```









<spotPython.spot.spot.Spot at 0x15d1db550>

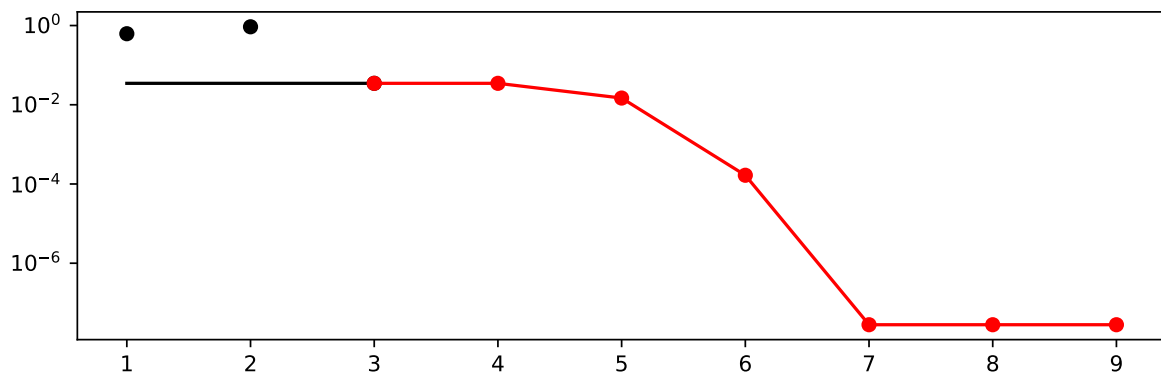
4.3.1 Results

```
spot_1.print_results()
```

```
min y: 2.7998468612116063e-08  
x0: -0.0001673274293477195
```

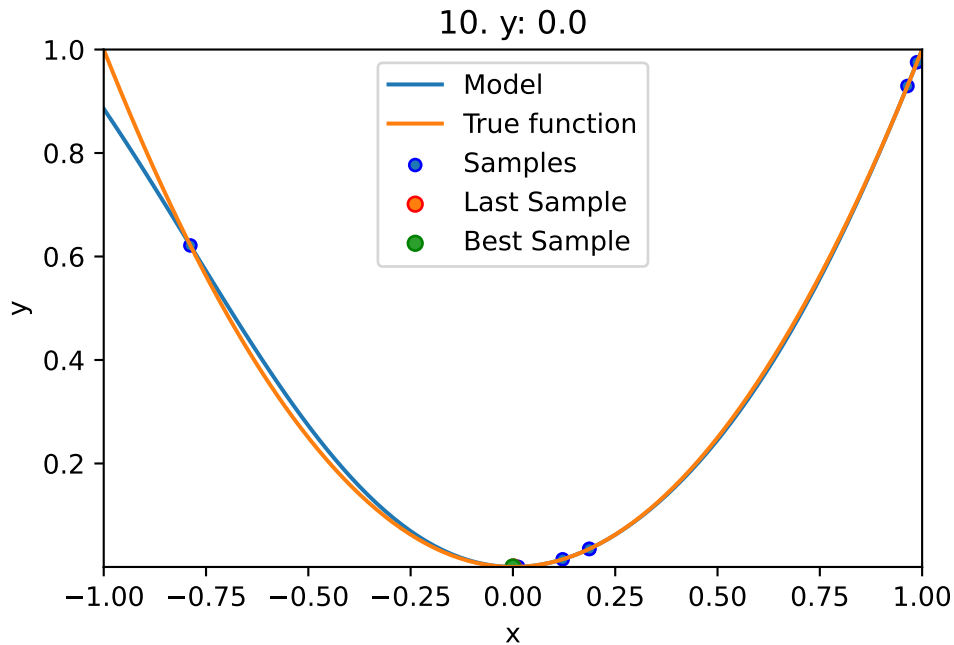
```
[['x0', -0.0001673274293477195]]
```

```
spot_1.plot_progress(log_y=True)
```



- The method `plot_model` plots the final surrogate:

```
spot_1.plot_model()
```

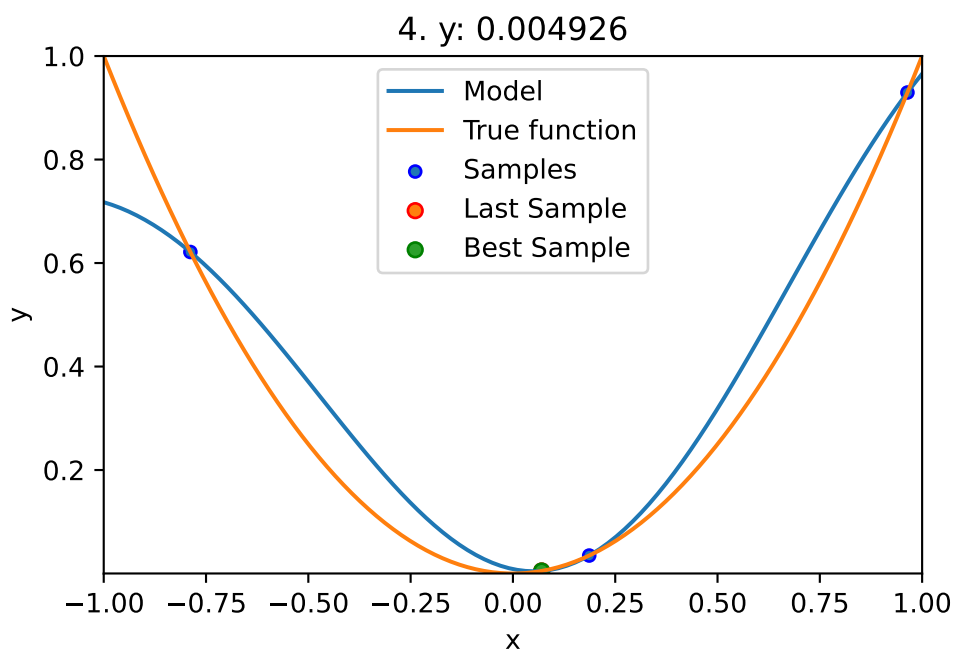
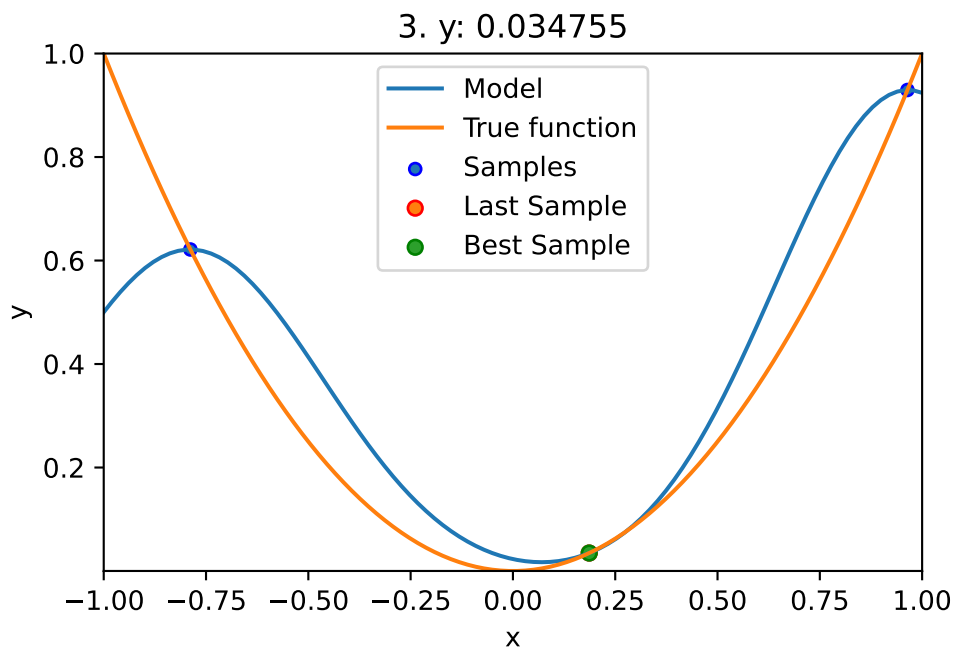


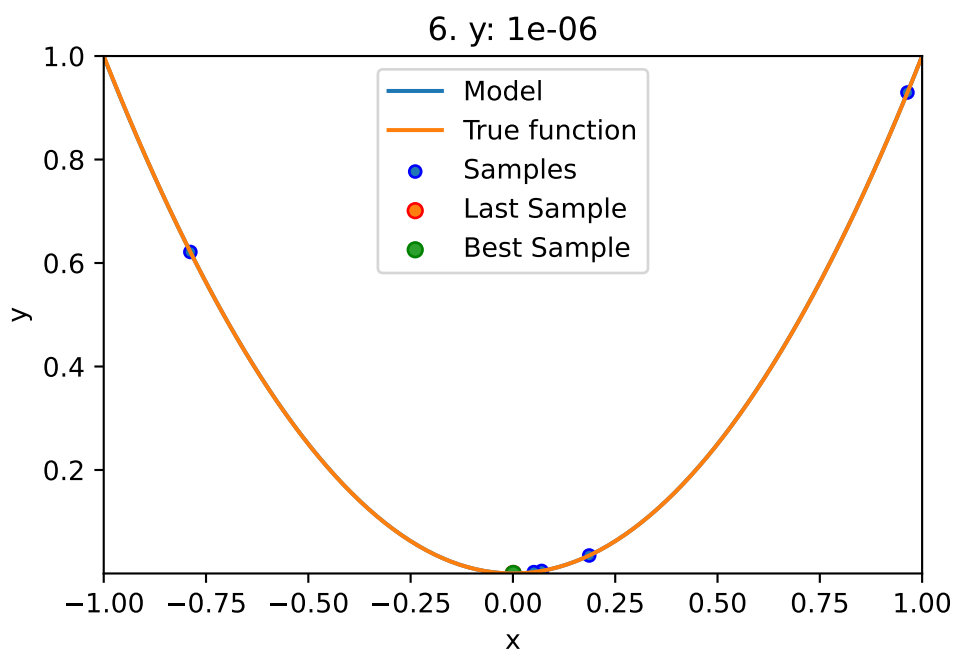
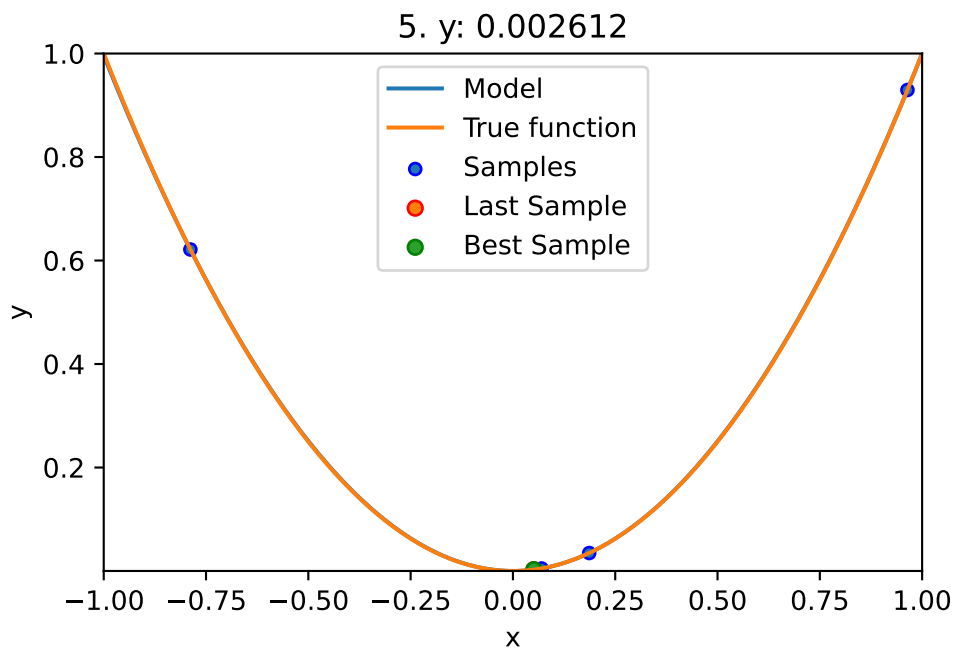
4.4 Example: Sklearn Model GaussianProcess

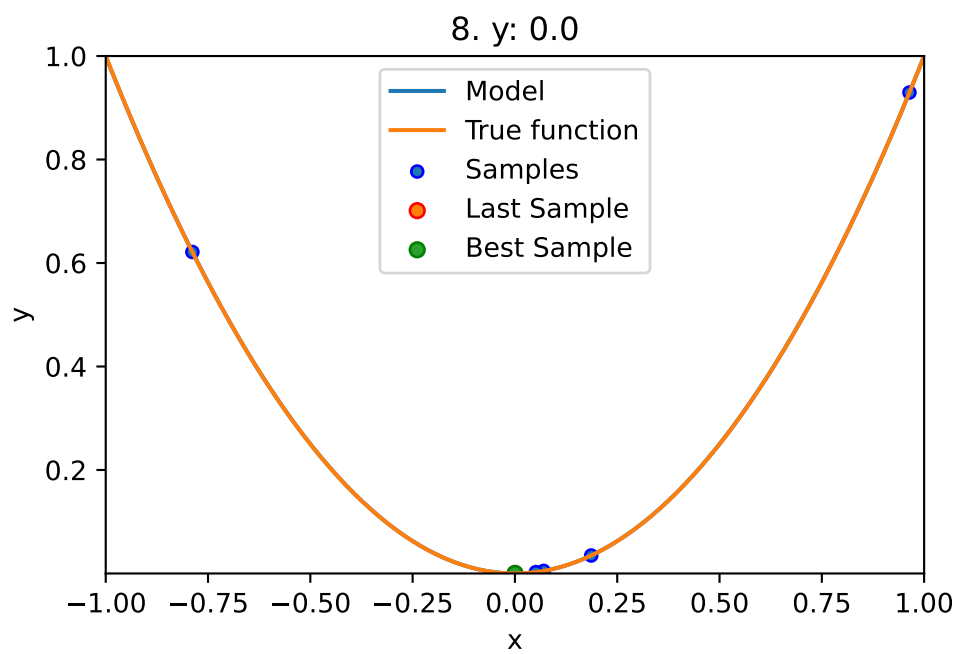
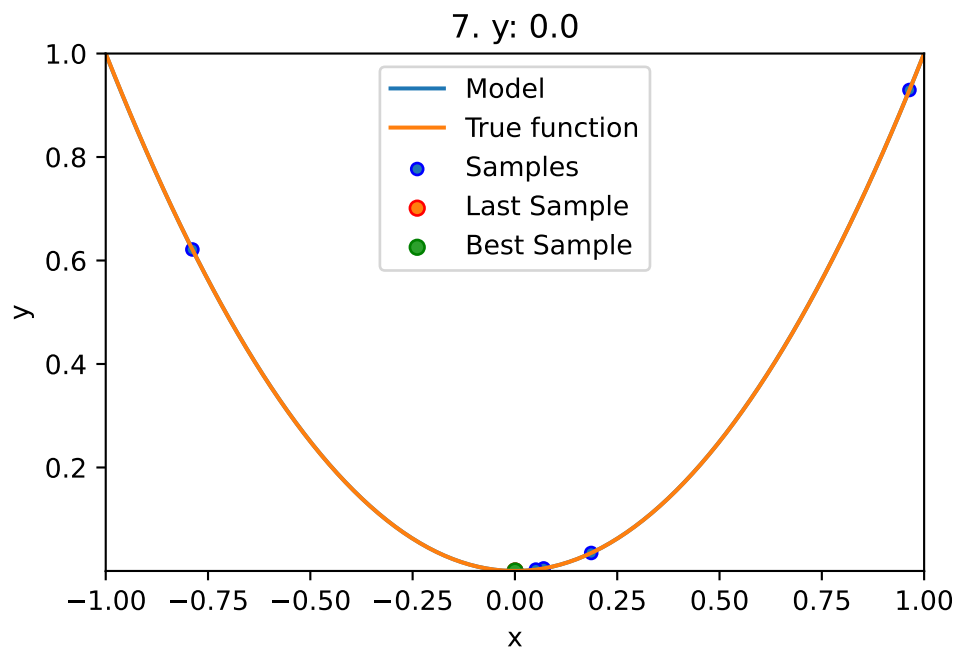
- This example visualizes the search process on the `GaussianProcessRegression` surrogate from `sklearn`.
- Therefore `surrogate = S_GP` is added to the argument list.

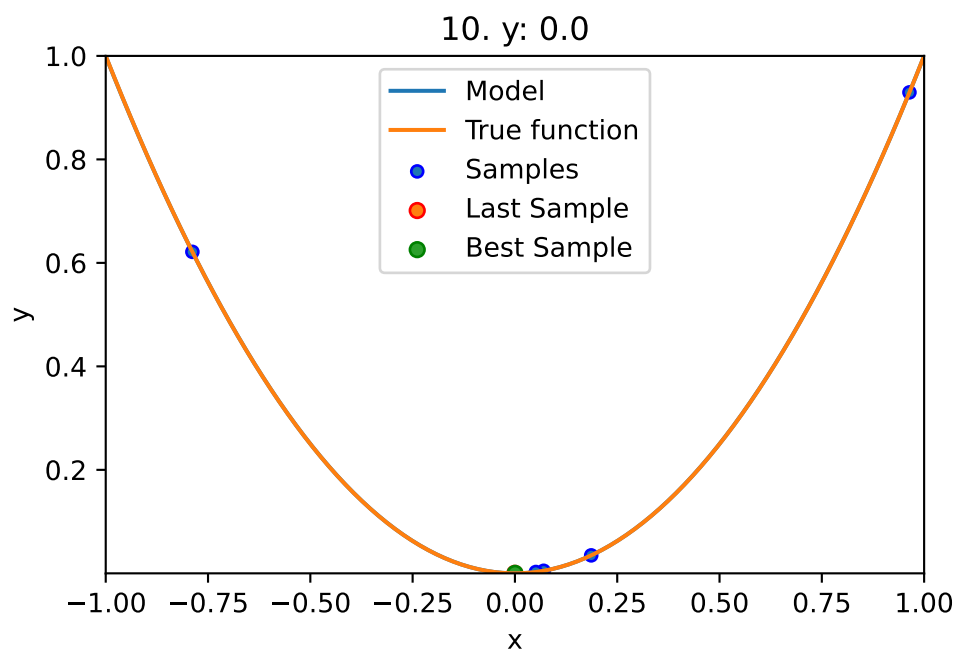
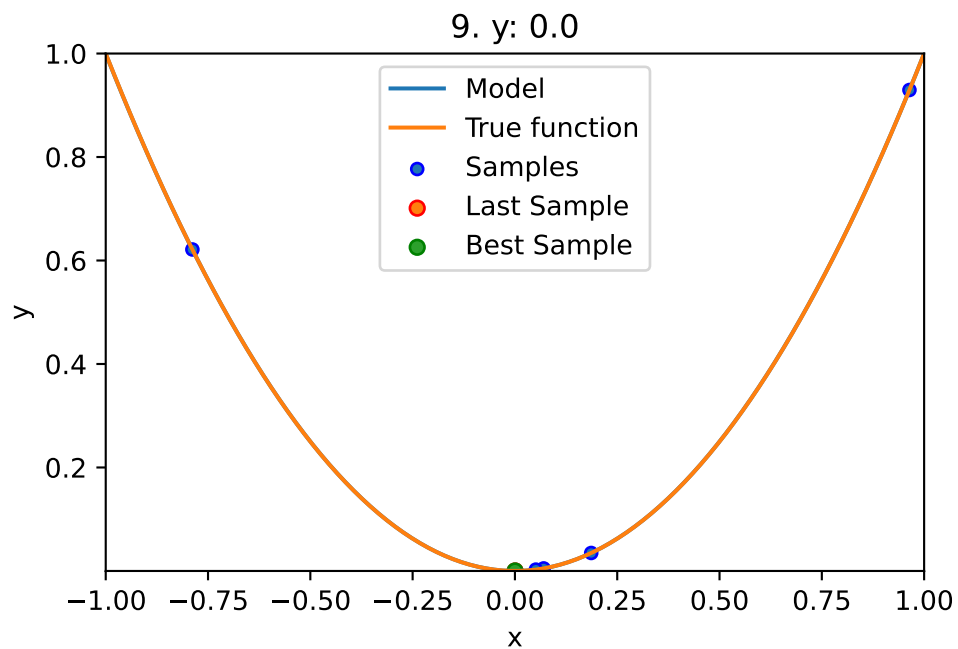
```
fun = analytical(seed=123).fun_sphere
spot_1_GP = spot.Spot(fun=fun,
                      lower = lower,
                      upper = upper,
                      fun_evals = 10,
                      max_time = inf,
                      seed=123,
                      show_models= True,
                      design_control={"init_size": 3},
                      surrogate = S_GP)

spot_1_GP.run()
```







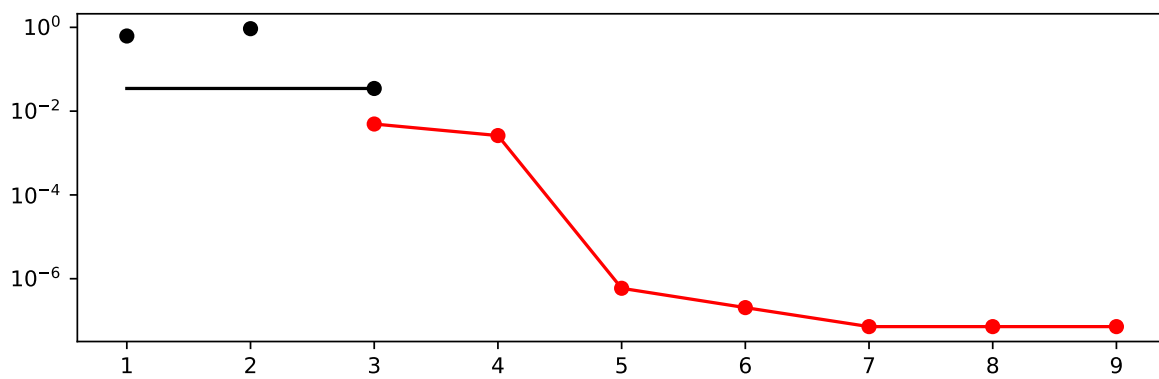
<spotPython.spot.spot.Spot at 0x15d0019c0>

```
spot_1_GP.print_results()
```

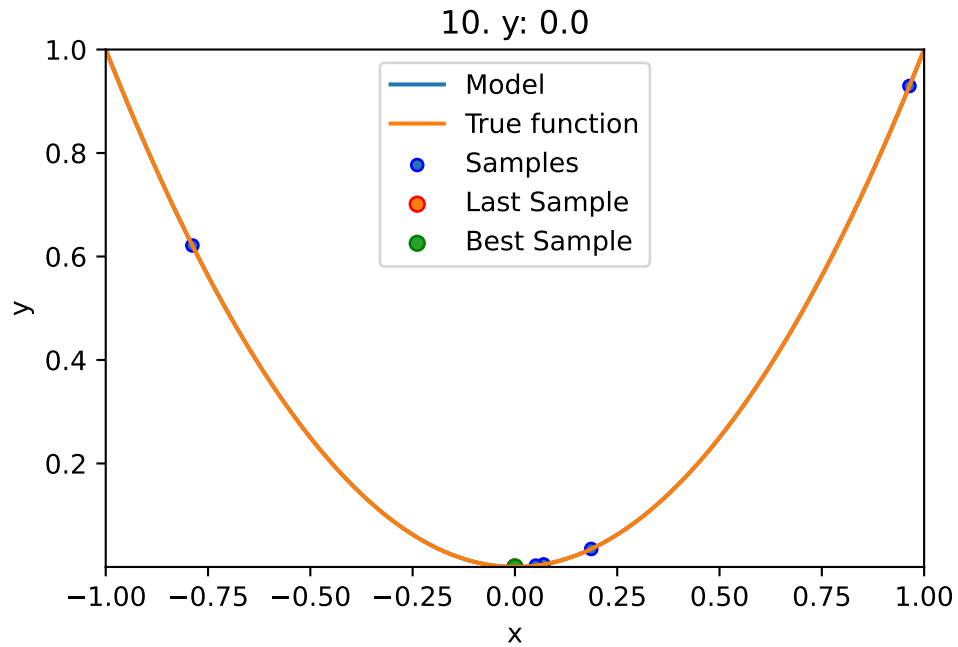
```
min y: 7.185357301597283e-08  
x0: 0.00026805516785910477
```

```
[['x0', 0.00026805516785910477]]
```

```
spot_1_GP.plot_progress(log_y=True)
```



```
spot_1_GP.plot_model()
```



4.5 Exercises

4.5.1 `DecisionTreeRegressor`

- Describe the surrogate model.
- Use the surrogate as the model for optimization.

4.5.2 `RandomForestRegressor`

- Describe the surrogate model.
- Use the surrogate as the model for optimization.

4.5.3 `linear_model.LinearRegression`

- Describe the surrogate model.
- Use the surrogate as the model for optimization.

4.5.4 `linear_model.Ridge`

- Describe the surrogate model.
- Use the surrogate as the model for optimization.

4.6 Exercise 2

- Compare the performance of the five different surrogates on both objective functions:
 - `spotPython`'s internal Kriging
 - `DecisionTreeRegressor`
 - `RandomForestRegressor`
 - `linear_model.LinearRegression`
 - `linear_model.Ridge`

5 Sequential Parameter Optimization: Using scipy Optimizers

This notebook describes how different optimizers from the `scipy optimize` package can be used on the surrogate. The optimization algorithms are available from <https://docs.scipy.org/doc/scipy/reference/optimize.html>

```
import numpy as np
from math import inf
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from scipy.optimize import shgo
from scipy.optimize import direct
from scipy.optimize import differential_evolution
from scipy.optimize import dual_annealing
from scipy.optimize import basinhopping
import matplotlib.pyplot as plt
```

5.1 The Objective Function Branin

- The `spotPython` package provides several classes of objective functions.
- We will use an analytical objective function, i.e., a function that can be described by a (closed) formula.
- Here we will use the Branin function. The 2-dim Branin function is

$$y = a * (x_2 - b * x_1^2 + c * x_1 - r)^2 + s * (1 - t) * \cos(x_1) + s,$$

where values of a , b , c , r , s and t are: $a = 1$, $b = 5.1/(4 * \pi^2)$, $c = 5/\pi$, $r = 6$, $s = 10$ and $t = 1/(8 * \pi)$.

- It has three global minima:

$$f(x) = 0.397887 \text{ at } (-\pi, 12.275), (\pi, 2.275), \text{ and } (9.42478, 2.475).$$

- Input Domain: This function is usually evaluated on the square x_1 in $[-5, 10]$ x x_2 in $[0, 15]$.

```
from spotPython.fun.objectivefunctions import analytical
lower = np.array([-5,-0])
upper = np.array([10,15])

fun = analytical(seed=123).fun_branin
```

5.2 The Optimizer

- Differential Evolution from the `scikit.optimize` package, see https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.differential_evolution.html#scipy.optimize.differential_evolution is the default optimizer for the search on the surrogate.

- Other optimizers that are available in `spotPython`:

- `dual_annealing`
- `direct`
- `shgo`
- `basinhopping`, see <https://docs.scipy.org/doc/scipy/reference/optimize.html#global-optimization>.

- These can be selected as follows:

```
surrogate_control = "model_optimizer": differential_evolution
```

- We will use `differential_evolution`.
- The optimizer can use 1000 evaluations. This value will be passed to the `differential_evolution` method, which has the argument `maxiter` (int). It defines the maximum number of generations over which the entire differential evolution population is evolved, see https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.differential_evolution.html#scipy.optimize.differential_evolution

```
spot_de = spot.Spot(fun=fun,
                    lower = lower,
                    upper = upper,
                    fun_evals = 20,
                    max_time = inf,
                    seed=125,
                    noise=False,
```

```

show_models= False,
design_control={"init_size": 10},
surrogate_control={"n_theta": 2,
                   "model_optimizer": differential_evolution,
                   "model_fun_evals": 1000,
                   })

spot_de.run()

```

<spotPython.spot.spot.Spot at 0x10c8741f0>

5.3 Print the Results

```
spot_de.print_results()
```

```

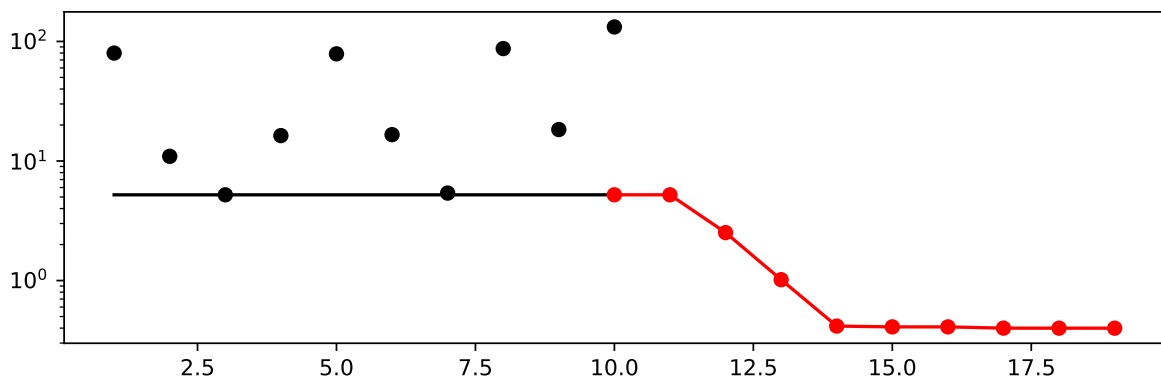
min y: 0.39955172035863207
x0: -3.1571400084867083
x1: 12.289948119375225

```

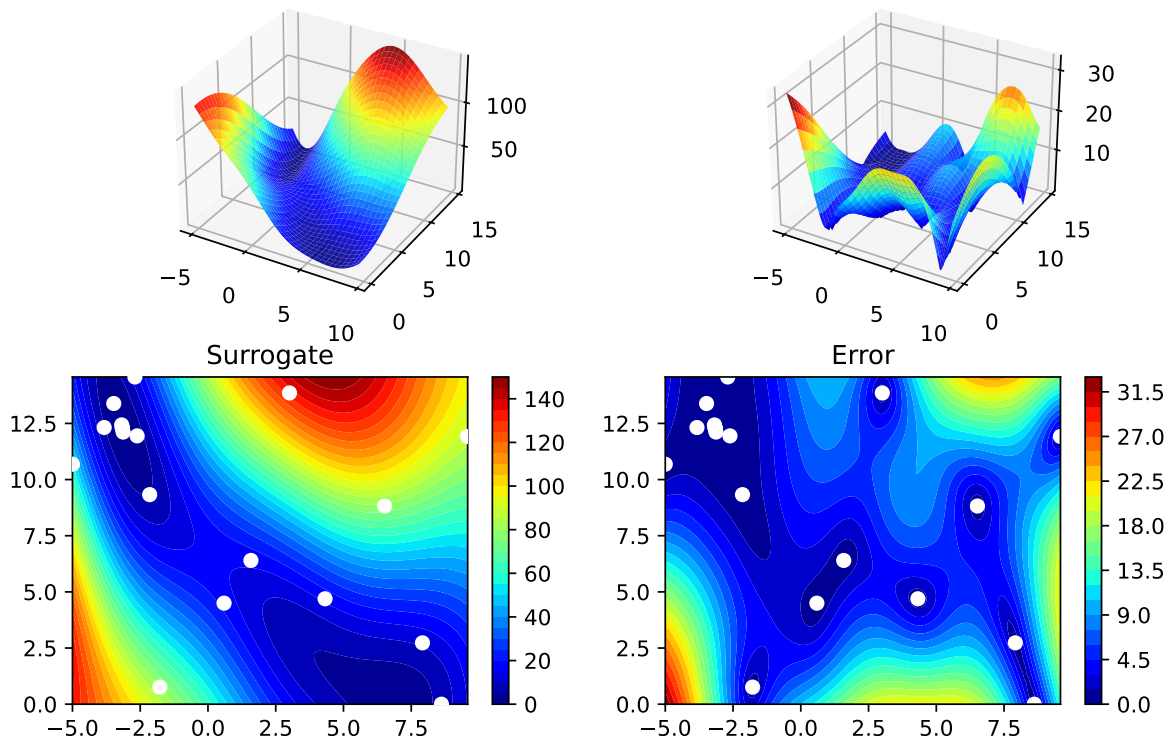
```
[['x0', -3.1571400084867083], ['x1', 12.289948119375225]]
```

5.4 Show the Progress

```
spot_de.plot_progress(log_y=True)
```



```
spot_de.surrogate.plot()
```



5.5 Exercises

5.5.1 dual_annealing

- Describe the optimization algorithm
- Use the algorithm as an optimizer on the surrogate

5.5.2 direct

- Describe the optimization algorithm
- Use the algorithm as an optimizer on the surrogate

5.5.3 shgo

- Describe the optimization algorithm
- Use the algorithm as an optimizer on the surrogate

5.5.4 basinhopping

- Describe the optimization algorithm
- Use the algorithm as an optimizer on the surrogate

5.5.5 Performance Comparison

Compare the performance and run time of the 5 different optimizers:

```
* `differential_evolution`  
* `dual_annealing`  
* `direct`  
* `shgo`  
* `basinhopping`.
```

The Branin function has three global minima:

- $f(x) = 0.397887$ at
 - $(-\pi, 12.275)$,
 - $(\pi, 2.275)$, and
 - $(9.42478, 2.475)$.
- Which optima are found by the optimizers? Does the `seed` change this behavior?

6 Sequential Parameter Optimization: Gaussian Process Models

- This notebook analyzes differences between
 - the Kriging implementation in `spotPython` and
 - the `GaussianProcessRegressor` in `scikit-learn`.

```
import numpy as np
from math import inf
from spotPython.fun.objectivefunctions import analytical
from spotPython.design.spacefilling import spacefilling
from spotPython.spot import spot
from spotPython.build.kriging import Kriging
from scipy.optimize import shgo
from scipy.optimize import direct
from scipy.optimize import differential_evolution
import matplotlib.pyplot as plt
import math as m
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF
```

6.1 Gaussian Processes Regression: Basic Introductory `scikit-learn` Example

- This is the example from `scikit-learn`: https://scikit-learn.org/stable/auto_examples/gaussian_process/plot_gpr.html
- After fitting our model, we see that the hyperparameters of the kernel have been optimized.
- Now, we will use our kernel to compute the mean prediction of the full dataset and plot the 95% confidence interval.

6.1.1 Train and Test Data

```
X = np.linspace(start=0, stop=10, num=1_000).reshape(-1, 1)
y = np.squeeze(X * np.sin(X))
rng = np.random.RandomState(1)
training_indices = rng.choice(np.arange(y.size), size=6, replace=False)
X_train, y_train = X[training_indices], y[training_indices]
```

6.1.2 Building the Surrogate With Sklearn

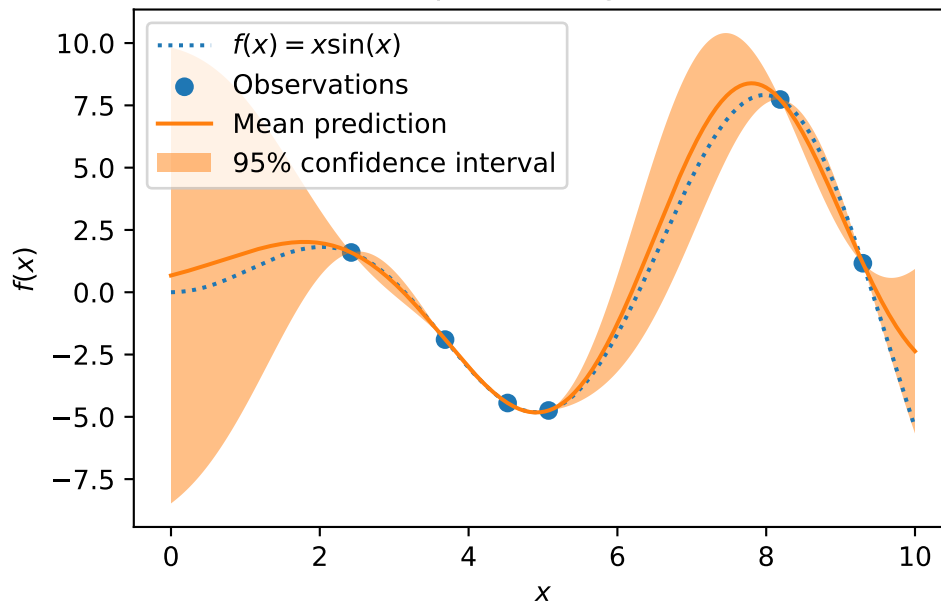
- The model building with `sklearn` consists of three steps:
 1. Instantiating the model, then
 2. fitting the model (using `fit`), and
 3. making predictions (using `predict`)

```
kernel = 1 * RBF(length_scale=1.0, length_scale_bounds=(1e-2, 1e2))
gaussian_process = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=9)
gaussian_process.fit(X_train, y_train)
mean_prediction, std_prediction = gaussian_process.predict(X, return_std=True)
```

6.1.3 Plotting the SklearnModel

```
plt.plot(X, y, label=r"$f(x) = x \sin(x)$", linestyle="dotted")
plt.scatter(X_train, y_train, label="Observations")
plt.plot(X, mean_prediction, label="Mean prediction")
plt.fill_between(
    X.ravel(),
    mean_prediction - 1.96 * std_prediction,
    mean_prediction + 1.96 * std_prediction,
    alpha=0.5,
    label=r"95% confidence interval",
)
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("sk-learn Version: Gaussian process regression on noise-free dataset")
```

sk-learn Version: Gaussian process regression on noise-free dataset



6.1.4 The spotPython Version

- The spotPython version is very similar:
 1. Instantiating the model, then
 2. fitting the model and
 3. making predictions (using `predict`).

```
S = Kriging(name='kriging', seed=123, log_level=50, cod_type="norm")
S.fit(X_train, y_train)
S_mean_prediction, S_std_prediction, S_ei = S.predict(X, return_val="all")
```

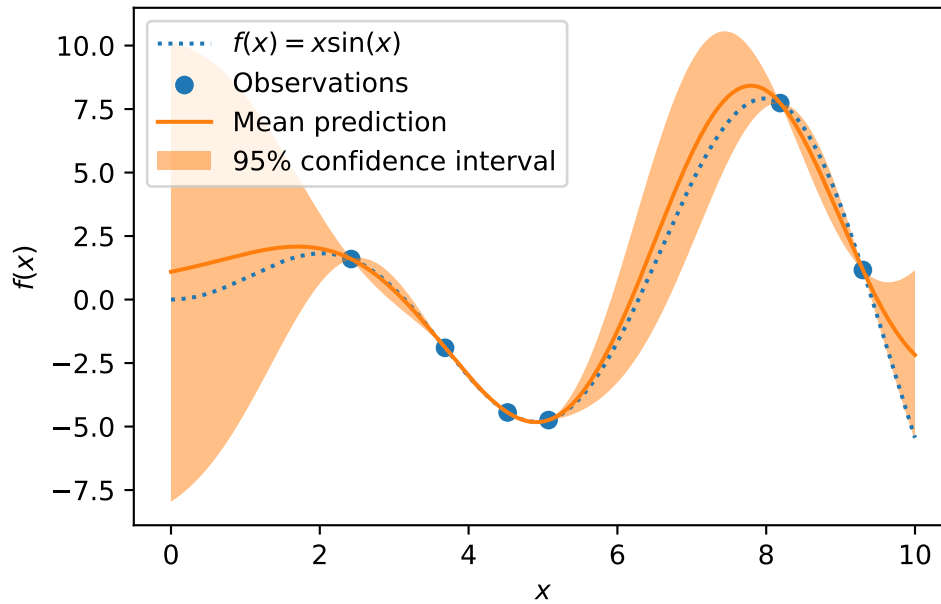
```
plt.plot(X, y, label=r"$f(x) = x \sin(x)$", linestyle="dotted")
plt.scatter(X_train, y_train, label="Observations")
plt.plot(X, S_mean_prediction, label="Mean prediction")
plt.fill_between(
    X.ravel(),
    S_mean_prediction - 1.96 * S_std_prediction,
    S_mean_prediction + 1.96 * S_std_prediction,
    alpha=0.5,
    label=r"95% confidence interval",
```

```

)
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("spotPython Version: Gaussian process regression on noise-free dataset")

```

spotPython Version: Gaussian process regression on noise-free dataset

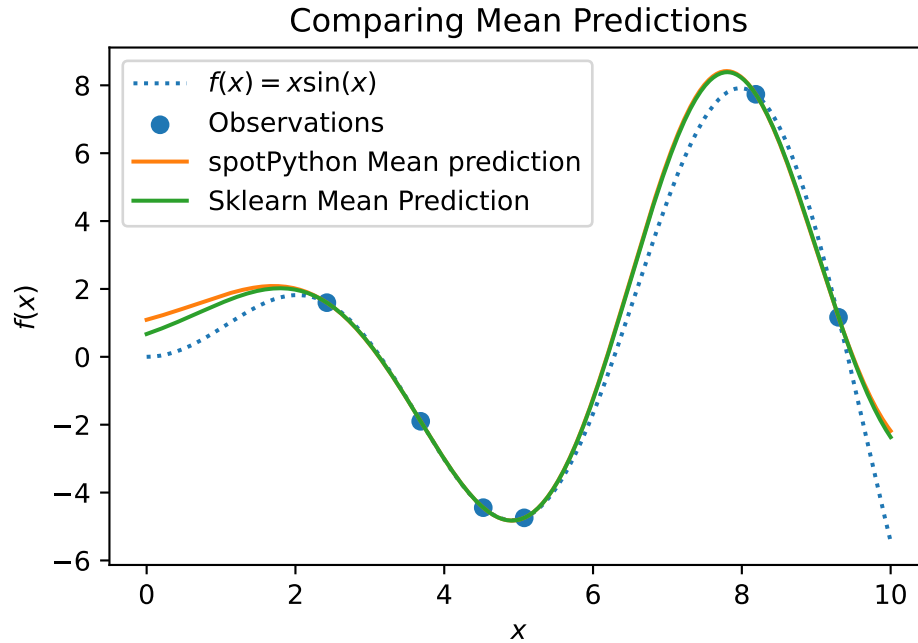


6.1.5 Visualizing the Differences Between the spotPython and the sklearn Model Fits

```

plt.plot(X, y, label=r"$f(x) = x \sin(x)$", linestyle="dotted")
plt.scatter(X_train, y_train, label="Observations")
plt.plot(X, S_mean_prediction, label="spotPython Mean prediction")
plt.plot(X, mean_prediction, label="Sklearn Mean Prediction")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Comparing Mean Predictions")

```

6.2 Exercises

6.2.1 Schonlau Example Function

- The Schonlau Example Function is based on sample points only (there is no analytical function description available):

```
X = np.linspace(start=0, stop=13, num=1_000).reshape(-1, 1)
X_train = np.array([1., 2., 3., 4., 12.]).reshape(-1,1)
y_train = np.array([0., -1.75, -2, -0.5, 5.])
```

- Describe the function.
- Compare the two models that were build using the `spotPython` and the `sklearn` surrogate.
- Note: Since there is no analytical function available, you might be interested in adding some points and describe the effects.

6.2.2 Forrester Example Function

- The Forrester Example Function is defined as follows:

$f(x) = (6x - 2)^2 \sin(12x - 4)$ for x in $[0, 1]$.

- Data points are generated as follows:

```
X = np.linspace(start=-0.5, stop=1.5, num=1_000).reshape(-1, 1)
X_train = np.array([0.0, 0.175, 0.225, 0.3, 0.35, 0.375, 0.5, 1]).reshape(-1, 1)
fun = analytical().fun_forrester
fun_control = {"sigma": 0.1,
               "seed": 123}
y = fun(X, fun_control=fun_control)
y_train = fun(X_train, fun_control=fun_control)
```

- Describe the function.
- Compare the two models that were build using the `spotPython` and the `sklearn` surrogate.
- Note: Modify the noise level ("`sigma`"), e.g., use a value of 0.2, and compare the two models.

```
fun_control = {"sigma": 0.2}
```

6.2.3 fun_runge Function (1-dim)

- The Runge function is defined as follows:

$f(x) = 1 / (1 + \sum(x_i))^2$

- Data points are generated as follows:

```
gen = spacefilling(1)
rng = np.random.RandomState(1)
lower = np.array([-10])
upper = np.array([10])
fun = analytical().fun_runge
fun_control = {"sigma": 0.025,
               "seed": 123}
X_train = gen.scipy_lhd(10, lower=lower, upper = upper).reshape(-1, 1)
y_train = fun(X, fun_control=fun_control)
X = np.linspace(start=-13, stop=13, num=1000).reshape(-1, 1)
y = fun(X, fun_control=fun_control)
```

- Describe the function.

- Compare the two models that were build using the `spotPython` and the `sklearn` surrogate.
- Note: Modify the noise level ("`sigma`"), e.g., use a value of 0.05, and compare the two models.

```
fun_control = {"sigma": 0.5}
```

6.2.4 fun_cubed (1-dim)

- The Cubed function is defined as follows:

```
np.sum(X[i]** 3)
```

- Data points are generated as follows:

```
gen = spacefilling(1)
rng = np.random.RandomState(1)
lower = np.array([-10])
upper = np.array([10])
fun = analytical().fun_cubed
fun_control = {"sigma": 0.025,
               "seed": 123}
X_train = gen.scipy_lhd(10, lower=lower, upper = upper).reshape(-1,1)
y_train = fun(X, fun_control=fun_control)
X = np.linspace(start=-13, stop=13, num=1000).reshape(-1, 1)
y = fun(X, fun_control=fun_control)
```

- Describe the function.
- Compare the two models that were build using the `spotPython` and the `sklearn` surrogate.
- Note: Modify the noise level ("`sigma`"), e.g., use a value of 0.05, and compare the two models.

```
fun_control = {"sigma": 0.05}
```

6.2.5 The Effect of Noise

How does the behavior of the `spotPython` fit changes when the argument `noise` is set to `True`, i.e.,

```
S = Kriging(name='kriging', seed=123, n_theta=1, noise=True)
```

is used?

7 Expected Improvement

7.1 Example: Spot and the 1-dim Sphere Function

```
import numpy as np
from math import inf
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from scipy.optimize import shgo
from scipy.optimize import direct
from scipy.optimize import differential_evolution
import matplotlib.pyplot as plt
```

7.1.1 The Objective Function: 1-dim Sphere

- The `spotPython` package provides several classes of objective functions.
- We will use an analytical objective function, i.e., a function that can be described by a (closed) formula:

$$f(x) = x^2$$

```
fun = analytical().fun_sphere
```

```
fun = analytical().fun_sphere
fun_control = {"sigma": 0,
               "seed": 123}
```

- The size of the `lower` bound vector determines the problem dimension.
- Here we will use `np.array([-1])`, i.e., a one-dim function.

```
spot_1 = spot.Spot(fun=fun,
                   lower = np.array([-1]),
                   upper = np.array([1]))
```

```
spot_1.run()
```

```
<spotPython.spot.spot.Spot at 0x15d852980>
```

7.1.2 Results

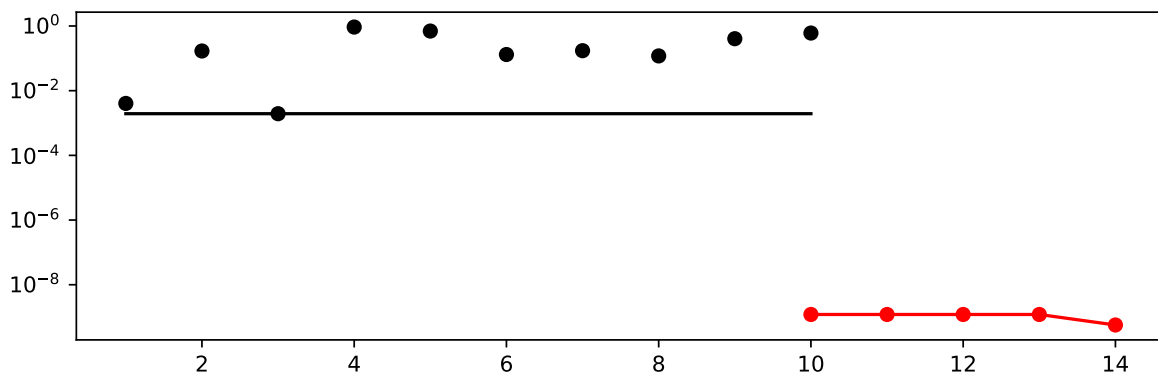
```
spot_1.print_results()
```

```
min y: 5.69019918867849e-10
```

```
x0: 2.3854138401288967e-05
```

```
[['x0', 2.3854138401288967e-05]]
```

```
spot_1.plot_progress(log_y=True)
```

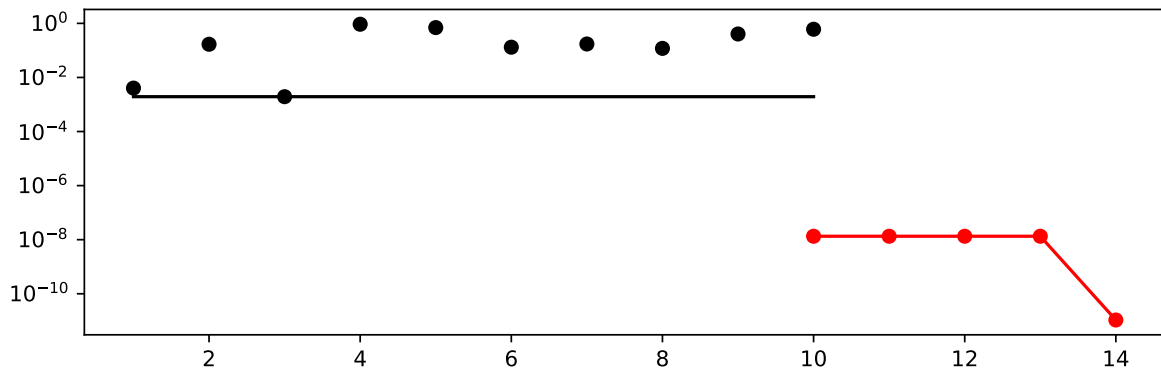


7.2 Same, but with EI as infill_criterion

```
spot_1_ei = spot.Spot(fun=fun,  
                      lower = np.array([-1]),  
                      upper = np.array([1]),  
                      infill_criterion = "ei")  
spot_1_ei.run()
```

```
<spotPython.spot.spot.Spot at 0x162316290>
```

```
spot_1_ei.plot_progress(log_y=True)
```



```
spot_1_ei.print_results()
```

```
min y: 1.0703048868228972e-11
```

```
x0: -3.271551446673118e-06
```

```
[['x0', -3.271551446673118e-06]]
```

7.3 Non-isotropic Kriging

```
spot_2_ei_noniso = spot.Spot(fun=fun,
                             lower = np.array([-1, -1]),
                             upper = np.array([1, 1]),
                             fun_evals = 20,
                             fun_repeats = 1,
                             max_time = inf,
                             noise = False,
                             tolerance_x = np.sqrt(np.spacing(1)),
                             var_type=["num"],
                             infill_criterion = "ei",
                             n_points = 1,
                             seed=123,
                             log_level = 50,
                             show_models=True,
                             fun_control = fun_control,
```

```

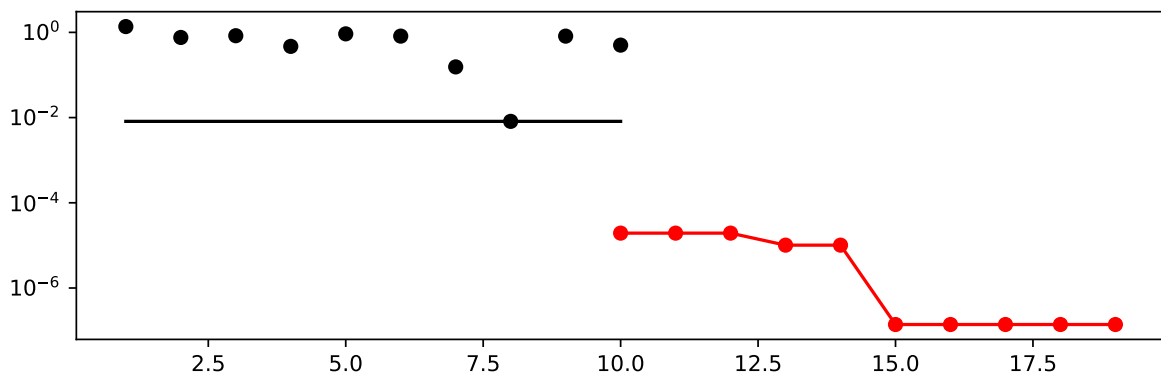
design_control={"init_size": 10,
               "repeats": 1},
surrogate_control={"noise": False,
                  "cod_type": "norm",
                  "min_theta": -4,
                  "max_theta": 3,
                  "n_theta": 2,
                  "model_optimizer": differential_evolution,
                  "model_fun_evals": 1000,
                  })

spot_2_ei_noniso.run()

```

<spotPython.spot.spot.Spot at 0x1624299c0>

```
spot_2_ei_noniso.plot_progress(log_y=True)
```



```
spot_2_ei_noniso.print_results()
```

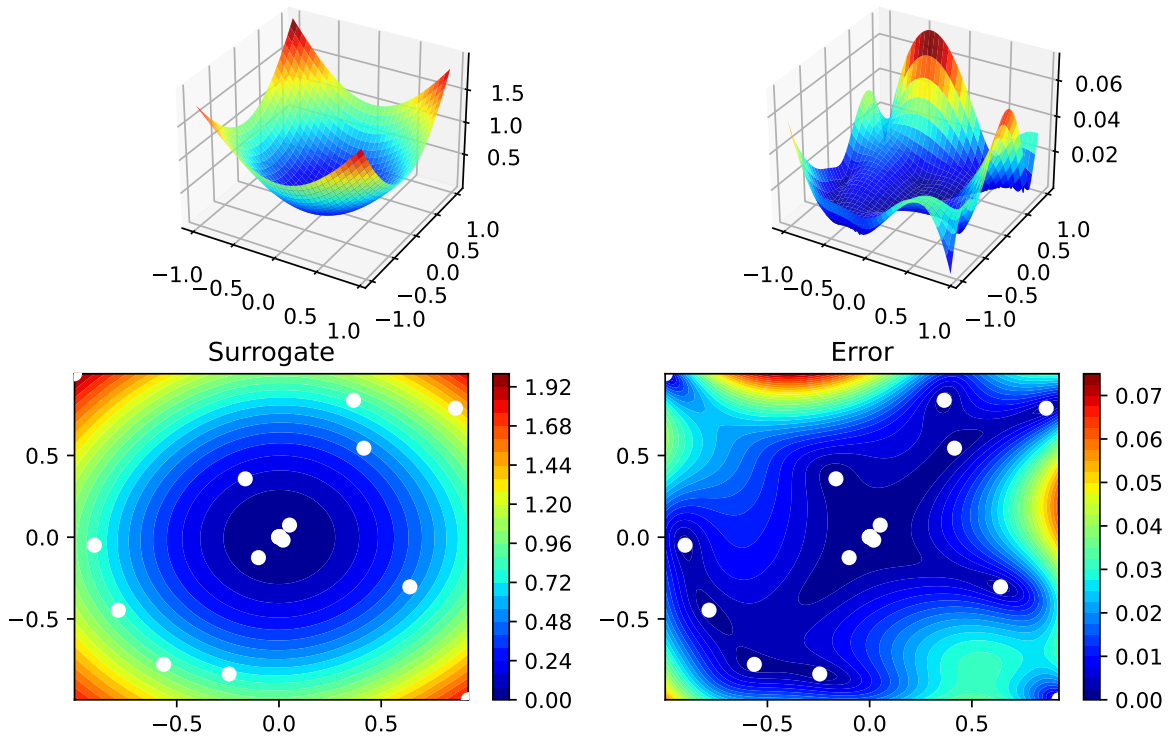
```

min y: 1.3904484958142096e-07
x0: -0.00024787994221983677
x1: 0.00027856845447126876

```

```
[['x0', -0.00024787994221983677], ['x1', 0.00027856845447126876]]
```

```
spot_2_ei_noniso.surrogate.plot()
```

7.4 Using sklearn Surrogates

7.4.1 The spot Loop

The `spot` loop consists of the following steps:

1. Init: Build initial design X
2. Evaluate initial design on real objective f : $y = f(X)$
3. Build surrogate: $S = S(X, y)$
4. Optimize on surrogate: $X_0 = \text{optimize}(S)$
5. Evaluate on real objective: $y_0 = f(X_0)$
6. Impute (Infill) new points: $X = X \cup X_0$, $y = y \cup y_0$.
7. Got 3.

The `spot` loop is implemented in R as follows:

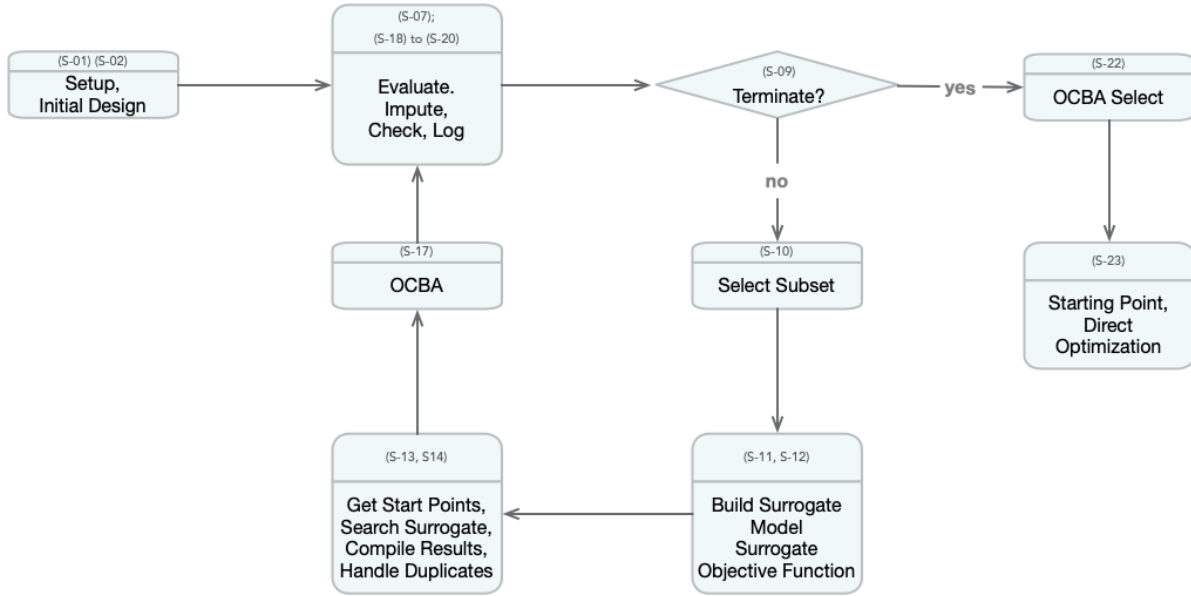


Figure 7.1: Visual representation of the model based search with SPOT. Taken from: Bartz-Beielstein, T., and Zaefferer, M. Hyperparameter tuning approaches. In Hyperparameter Tuning for Machine and Deep Learning with R - A Practical Guide, E. Bartz, T. Bartz-Beielstein, M. Zaefferer, and O. Mersmann, Eds. Springer, 2022, ch. 4, pp. 67–114.

7.4.2 spot: The Initial Model

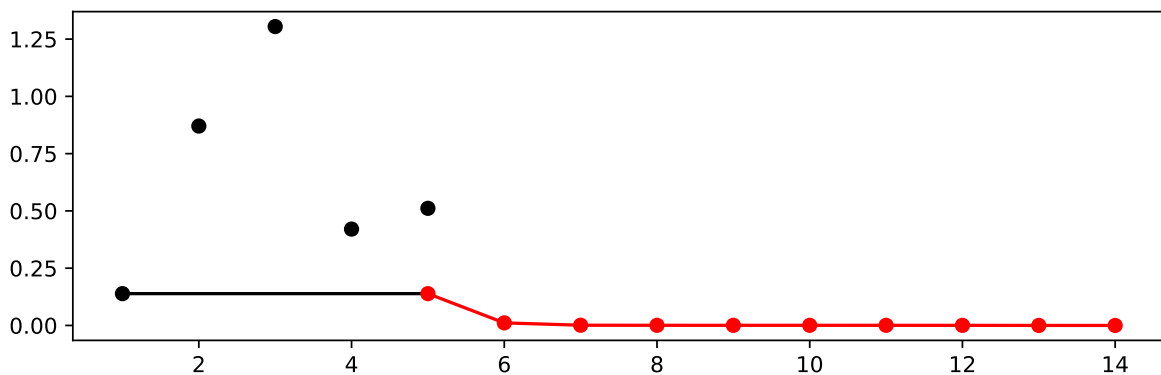
7.4.2.1 Example: Modifying the initial design size

This is the “Example: Modifying the initial design size” from Chapter 4.5.1 in [bart21i].

```
spot_ei = spot.Spot(fun=fun,  
                    lower = np.array([-1,-1]),  
                    upper= np.array([1,1]),  
                    design_control={"init_size": 5})  
spot_ei.run()
```

<spotPython.spot.spot.Spot at 0x1626bde70>

```
spot_ei.plot_progress()
```



```
np.min(spot_1.y), np.min(spot_ei.y)
```

(5.69019918867849e-10, 1.992607881423438e-05)

7.4.3 Init: Build Initial Design

```
from spotPython.design.spacefilling import spacefilling  
from spotPython.build.kriging import Kriging  
from spotPython.fun.objectivefunctions import analytical  
gen = spacefilling(2)
```

```

rng = np.random.RandomState(1)
lower = np.array([-5,-0])
upper = np.array([10,15])
fun = analytical().fun_branin
fun_control = {"sigma": 0,
               "seed": 123}

X = gen.scipy_lhd(10, lower=lower, upper = upper)
print(X)
y = fun(X, fun_control=fun_control)
print(y)

```

```

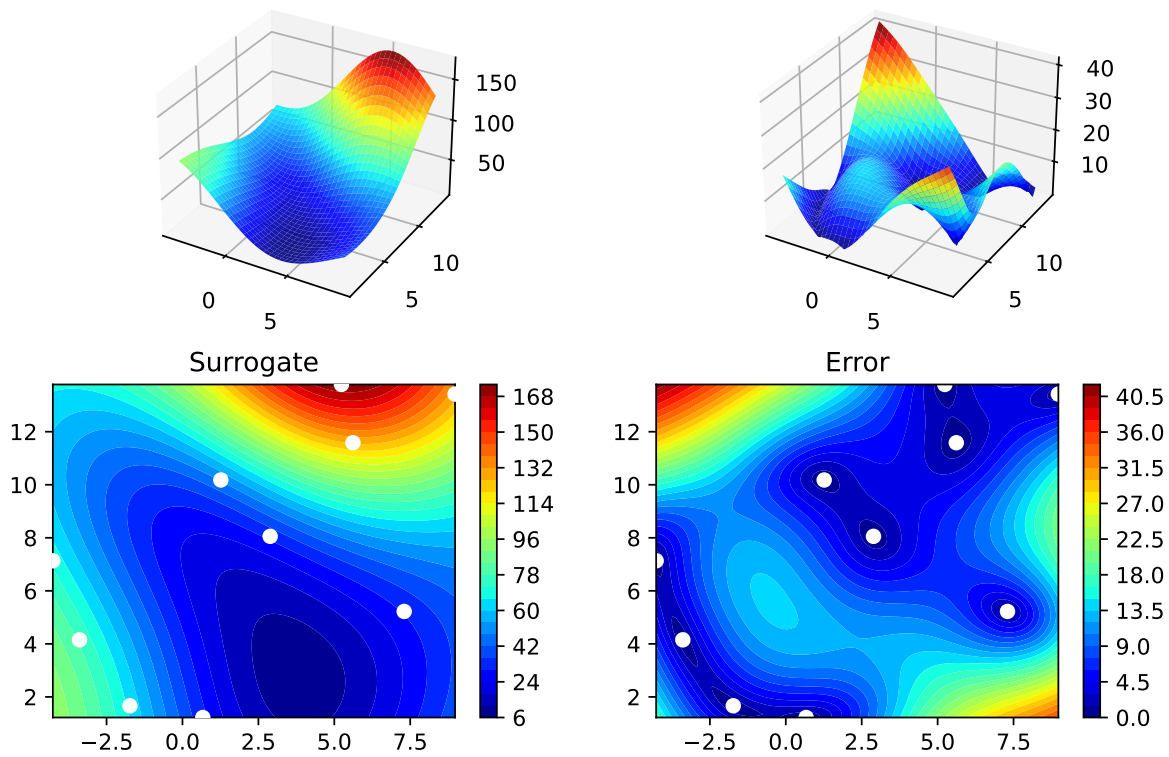
[[ 8.97647221 13.41926847]
 [ 0.66946019  1.22344228]
 [ 5.23614115 13.78185824]
 [ 5.6149825  11.5851384 ]
 [-1.72963184  1.66516096]
 [-4.26945568  7.1325531 ]
 [ 1.26363761 10.17935555]
 [ 2.88779942  8.05508969]
 [-3.39111089  4.15213772]
 [ 7.30131231  5.22275244]]
[128.95676449  31.73474356 172.89678121 126.71295908  64.34349975
 70.16178611  48.71407916  31.77322887  76.91788181  30.69410529]

```

```

S = Kriging(name='kriging', seed=123)
S.fit(X, y)
S.plot()

```



```

gen = spacefilling(2, seed=123)
X0 = gen.scipy_lhd(3)
gen = spacefilling(2, seed=345)
X1 = gen.scipy_lhd(3)
X2 = gen.scipy_lhd(3)
gen = spacefilling(2, seed=123)
X3 = gen.scipy_lhd(3)
X0, X1, X2, X3

```

```

(array([[0.77254938, 0.31539299],
        [0.59321338, 0.93854273],
        [0.27469803, 0.3959685 ]]),
array([[0.78373509, 0.86811887],
        [0.06692621, 0.6058029 ],
        [0.41374778, 0.00525456]]),
array([[0.121357 , 0.69043832],
        [0.41906219, 0.32838498],
        [0.86742658, 0.52910374]]),

```

```
array([[0.77254938, 0.31539299],
       [0.59321338, 0.93854273],
       [0.27469803, 0.3959685 ]])
```

7.4.4 Evaluate

7.4.5 Build Surrogate

7.4.6 A Simple Predictor

The code below shows how to use a simple model for prediction.

- Assume that only two (very costly) measurements are available:
 1. $f(0) = 0.5$
 2. $f(2) = 2.5$
- We are interested in the value at $x_0 = 1$, i.e., $f(x_0 = 1)$, but cannot run an additional, third experiment.

```
from sklearn import linear_model
X = np.array([[0], [2]])
y = np.array([0.5, 2.5])
S_lm = linear_model.LinearRegression()
S_lm = S_lm.fit(X, y)
X0 = np.array([[1]])
y0 = S_lm.predict(X0)
print(y0)
```

[1.5]

- Central Idea:
 - Evaluation of the surrogate model `S_lm` is much cheaper (or / and much faster) than running the real-world experiment f .

7.5 Gaussian Processes regression: basic introductory example

This example was taken from [scikit-learn](#). After fitting our model, we see that the hyperparameters of the kernel have been optimized. Now, we will use our kernel to compute the mean prediction of the full dataset and plot the 95% confidence interval.

```

import numpy as np
import matplotlib.pyplot as plt
import math as m
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF

X = np.linspace(start=0, stop=10, num=1_000).reshape(-1, 1)
y = np.squeeze(X * np.sin(X))
rng = np.random.RandomState(1)
training_indices = rng.choice(np.arange(y.size), size=6, replace=False)
X_train, y_train = X[training_indices], y[training_indices]

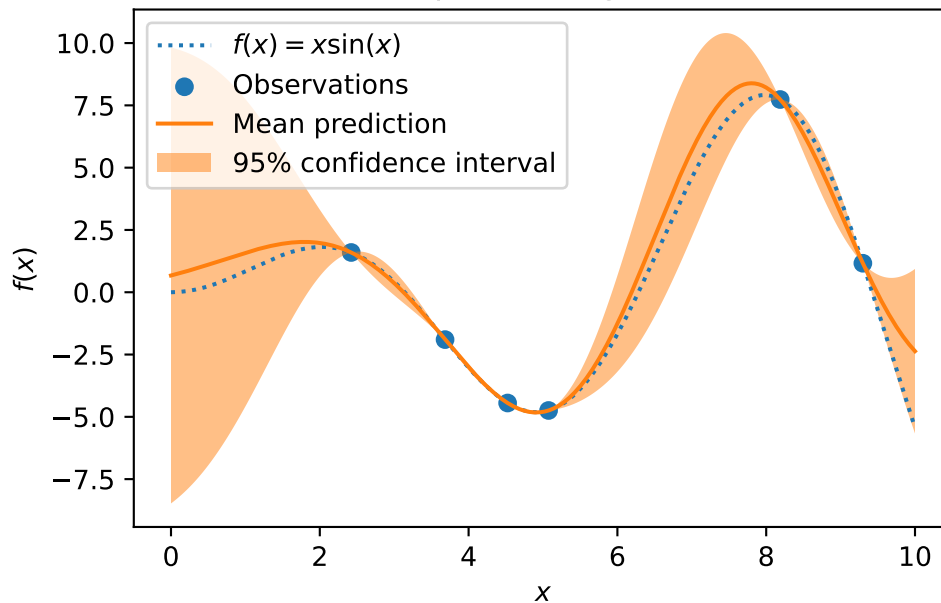
kernel = 1 * RBF(length_scale=1.0, length_scale_bounds=(1e-2, 1e2))
gaussian_process = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=9)
gaussian_process.fit(X_train, y_train)
gaussian_process.kernel_

mean_prediction, std_prediction = gaussian_process.predict(X, return_std=True)

plt.plot(X, y, label=r"$f(x) = x \sin(x)$", linestyle="dotted")
plt.scatter(X_train, y_train, label="Observations")
plt.plot(X, mean_prediction, label="Mean prediction")
plt.fill_between(
    X.ravel(),
    mean_prediction - 1.96 * std_prediction,
    mean_prediction + 1.96 * std_prediction,
    alpha=0.5,
    label=r"95% confidence interval",
)
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("sk-learn Version: Gaussian process regression on noise-free dataset")

```

sk-learn Version: Gaussian process regression on noise-free dataset



```
from spotPython.build.kriging import Kriging
import numpy as np
import matplotlib.pyplot as plt
rng = np.random.RandomState(1)
X = np.linspace(start=0, stop=10, num=1_000).reshape(-1, 1)
y = np.squeeze(X * np.sin(X))
training_indices = rng.choice(np.arange(y.size), size=6, replace=False)
X_train, y_train = X[training_indices], y[training_indices]

S = Kriging(name='kriging', seed=123, log_level=50, cod_type="norm")
S.fit(X_train, y_train)

mean_prediction, std_prediction, ei = S.predict(X, return_val="all")

std_prediction

plt.plot(X, y, label=r"$f(x) = x \sin(x)$", linestyle="dotted")
plt.scatter(X_train, y_train, label="Observations")
plt.plot(X, mean_prediction, label="Mean prediction")
plt.fill_between(
```

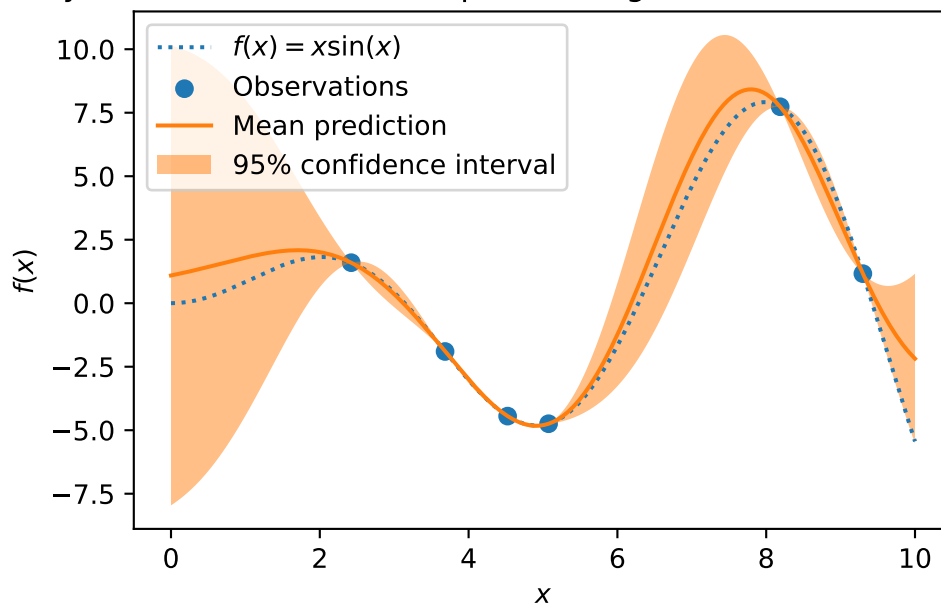


```

X.ravel(),
mean_prediction - 1.96 * std_prediction,
mean_prediction + 1.96 * std_prediction,
alpha=0.5,
label=r"95% confidence interval",
)
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("spotPython Version: Gaussian process regression on noise-free dataset")

```

spotPython Version: Gaussian process regression on noise-free dataset



7.6 The Surrogate: Using scikit-learn models

Default is the internal `kriging` surrogate.

```
S_0 = Kriging(name='kriging', seed=123)
```

Models from `scikit-learn` can be selected, e.g., Gaussian Process:

```
# Needed for the sklearn surrogates:
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn import linear_model
from sklearn import tree
import pandas as pd

kernel = 1 * RBF(length_scale=1.0, length_scale_bounds=(1e-2, 1e2))
S_GP = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=9)
```

- and many more:

```
S_Tree = DecisionTreeRegressor(random_state=0)
S_LM = linear_model.LinearRegression()
S_Ridge = linear_model.Ridge()
S_RF = RandomForestRegressor(max_depth=2, random_state=0)
```

- The scikit-learn GP model S_GP is selected.

```
S = S_GP
```

```
isinstance(S, GaussianProcessRegressor)
```

True

```
from spotPython.fun.objectivefunctions import analytical
fun = analytical().fun_branin
lower = np.array([-5,-0])
upper = np.array([10,15])
design_control={"init_size": 5}
surrogate_control={
    "infill_criterion": None,
    "n_points": 1,
}
spot_GP = spot.Spot(fun=fun, lower = lower, upper= upper, surrogate=S,
    fun_evals = 15, noise = False, log_level = 50,
    design_control=design_control,
    surrogate_control=surrogate_control)
```

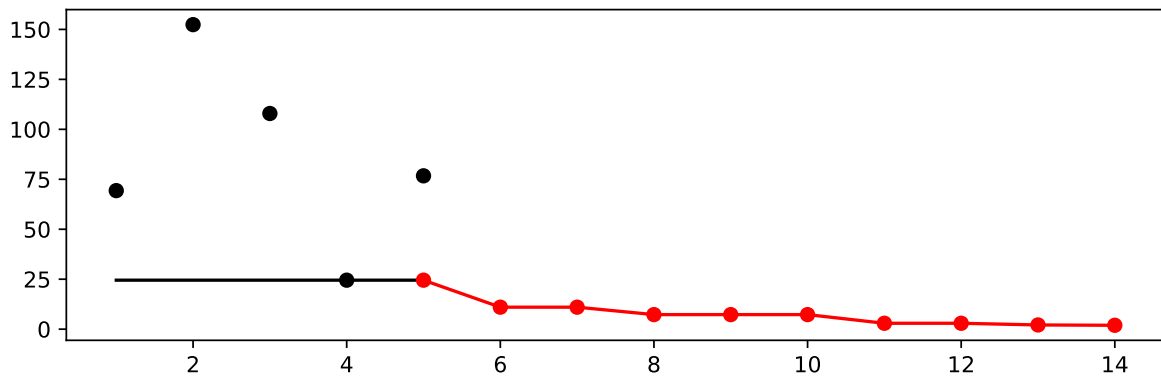
```
spot_GP.run()
```

```
<spotPython.spot.spot.Spot at 0x16296b7c0>
```

```
spot_GP.y
```

```
array([ 69.32459936, 152.38491454, 107.92560483,  24.51465459,  
       76.73500031,  86.30425705,  11.00311205,  16.11744353,  
        7.2811052 , 21.82320514,  10.96088904,   2.95188006,  
        3.02908492,   2.10496928,   1.94315642])
```

```
spot_GP.plot_progress()
```



```
spot_GP.print_results()
```

```
min y: 1.9431564172819389  
x0: 10.0  
x1: 2.998987424438876
```

```
[['x0', 10.0], ['x1', 2.998987424438876]]
```

7.7 Additional Examples

```

# Needed for the sklearn surrogates:
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn import linear_model
from sklearn import tree
import pandas as pd

kernel = 1 * RBF(length_scale=1.0, length_scale_bounds=(1e-2, 1e2))
S_GP = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=9)

from spotPython.build.kriging import Kriging
import numpy as np
import spotPython
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot

S_K = Kriging(name='kriging',
              seed=123,
              log_level=50,
              infill_criterion = "y",
              n_theta=1,
              noise=False,
              cod_type="norm")
fun = analytical().fun_sphere
lower = np.array([-1,-1])
upper = np.array([1,1])

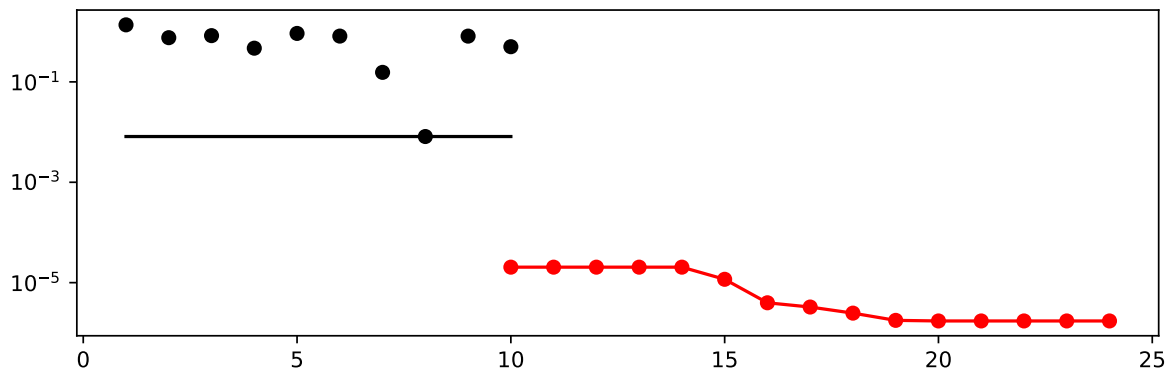
design_control={"init_size": 10}
surrogate_control={
    "n_points": 1,
}
spot_S_K = spot.Spot(fun=fun,
                    lower = lower,
                    upper= upper,
                    surrogate=S_K,
                    fun_evals = 25,
                    noise = False,
                    log_level = 50,

```

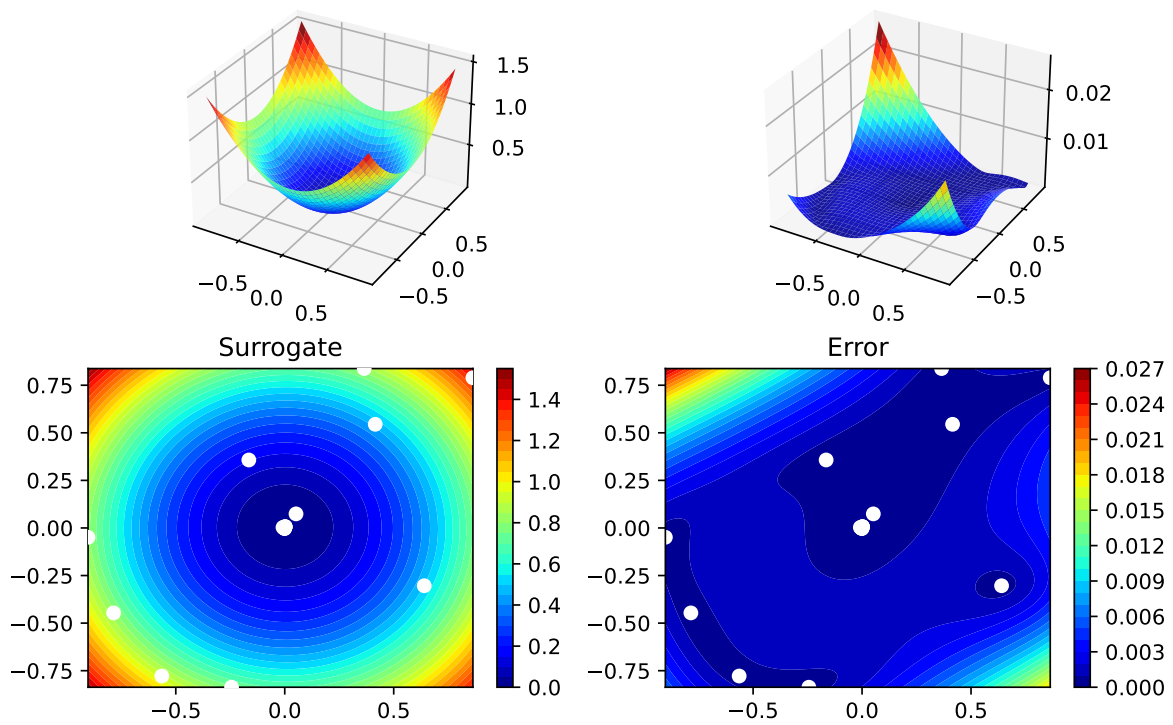
```
design_control=design_control,  
surrogate_control=surrogate_control)  
  
spot_S_K.run()
```

```
<spotPython.spot.spot.Spot at 0x162952830>
```

```
spot_S_K.plot_progress(log_y=True)
```



```
spot_S_K.surrogate.plot()
```



```
spot_S_K.print_results()
```

```
min y: 1.724871809162595e-06
x0: -0.001300204548042376
x1: 0.00018531039477729297
```

```
[['x0', -0.001300204548042376], ['x1', 0.00018531039477729297]]
```

7.7.1 Optimize on Surrogate

7.7.2 Evaluate on Real Objective

7.7.3 Impute / Infill new Points

7.8 Tests

```
import numpy as np
from spotPython.spot import spot
from spotPython.fun.objectivefunctions import analytical

fun_sphere = analytical().fun_sphere
spot_1 = spot.Spot(
    fun=fun_sphere,
    lower=np.array([-1, -1]),
    upper=np.array([1, 1]),
    n_points = 2
)

# (S-2) Initial Design:
spot_1.X = spot_1.design.scipy_lhd(
    spot_1.design_control["init_size"], lower=spot_1.lower, upper=spot_1.upper
)
print(spot_1.X)

# (S-3): Eval initial design:
spot_1.y = spot_1.fun(spot_1.X)
print(spot_1.y)

spot_1.surrogate.fit(spot_1.X, spot_1.y)
X0 = spot_1.suggest_new_X()
print(X0)
assert X0.size == spot_1.n_points * spot_1.k
```

```
[[ 0.86352963  0.7892358 ]
 [-0.24407197 -0.83687436]
 [ 0.36481882  0.8375811 ]
 [ 0.415331    0.54468512]
 [-0.56395091 -0.77797854]
 [-0.90259409 -0.04899292]]
```

```

[-0.16484832  0.35724741]
[ 0.05170659  0.07401196]
[-0.78548145 -0.44638164]
[ 0.64017497 -0.30363301]]
[1.36857656 0.75992983 0.83463487 0.46918172 0.92329124 0.8170764
 0.15480068 0.00815134 0.81623768 0.502017 ]

[[0.0015037  0.00422729]
 [0.0015037  0.00422729]]

```

7.9 EI: The Famous Schonlau Example

```

X_train0 = np.array([1, 2, 3, 4, 12]).reshape(-1,1)
X_train = np.linspace(start=0, stop=10, num=5).reshape(-1, 1)

from spotPython.build.kriging import Kriging
import numpy as np
import matplotlib.pyplot as plt

X_train = np.array([1., 2., 3., 4., 12.]).reshape(-1,1)
y_train = np.array([0., -1.75, -2, -0.5, 5.])

S = Kriging(name='kriging', seed=123, log_level=50, n_theta=1, noise=False, cod_type="non")
S.fit(X_train, y_train)

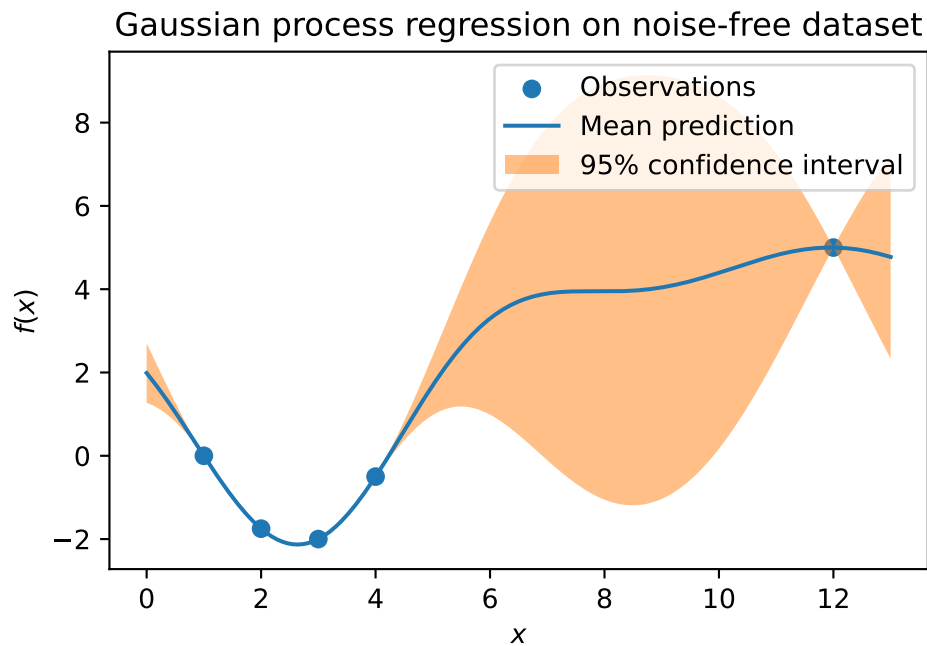
X = np.linspace(start=0, stop=13, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S.predict(X, return_val="all")

plt.scatter(X_train, y_train, label="Observations")
plt.plot(X, mean_prediction, label="Mean prediction")
if True:
    plt.fill_between(
        X.ravel(),
        mean_prediction - 2 * std_prediction,
        mean_prediction + 2 * std_prediction,
        alpha=0.5,
        label=r"95% confidence interval",
    )

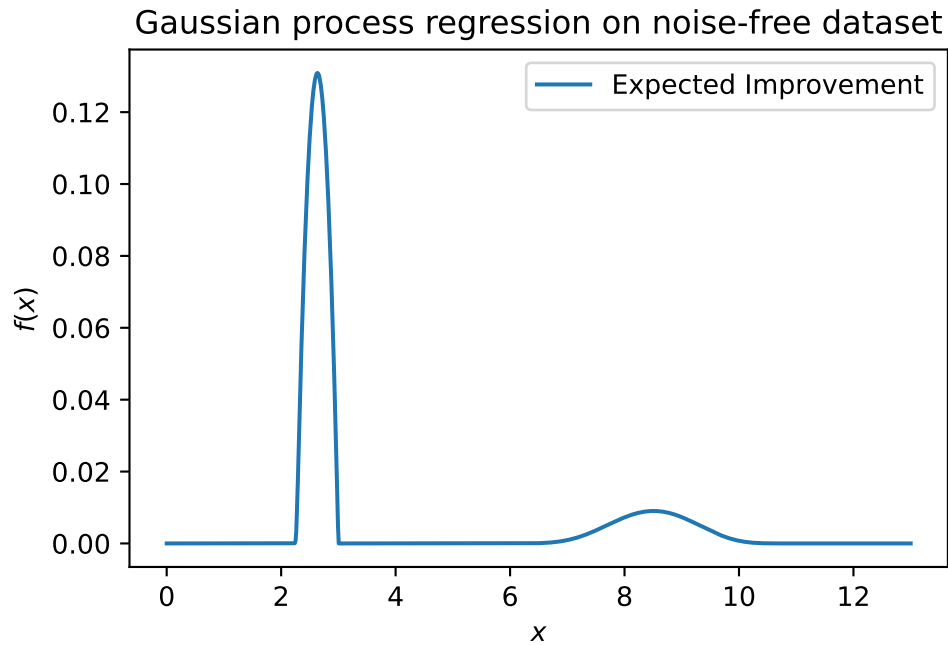
```



```
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Gaussian process regression on noise-free dataset")
```



```
#plt.plot(X, y, label=r"$f(x) = x \sin(x)$", linestyle="dotted")
# plt.scatter(X_train, y_train, label="Observations")
plt.plot(X, -ei, label="Expected Improvement")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Gaussian process regression on noise-free dataset")
```



S.log

```
{'negLnLike': array([1.20788205]),
 'theta': array([1.09276015]),
 'p': array([2.]),
 'Lambda': array([None], dtype=object)}
```

7.10 EI: The Forrester Example

```
from spotPython.build.kriging import Kriging
import numpy as np
import matplotlib.pyplot as plt
import spotPython
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot

# exact x locations are unknown:
X_train = np.array([0.0, 0.175, 0.225, 0.3, 0.35, 0.375, 0.5, 1]).reshape(-1,1)
```

```

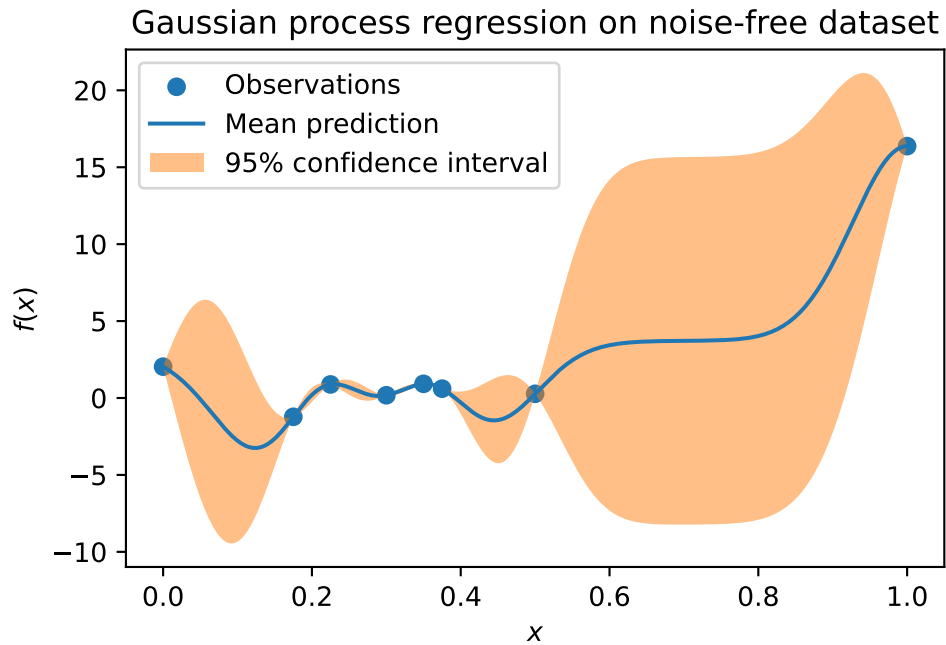
fun = analytical().fun_forrester
fun_control = {"sigma": 1.0,
               "seed": 123}
y_train = fun(X_train, fun_control=fun_control)

S = Kriging(name='kriging', seed=123, log_level=50, n_theta=1, noise=False, cod_type="normal")
S.fit(X_train, y_train)

X = np.linspace(start=0, stop=1, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S.predict(X, return_val="all")

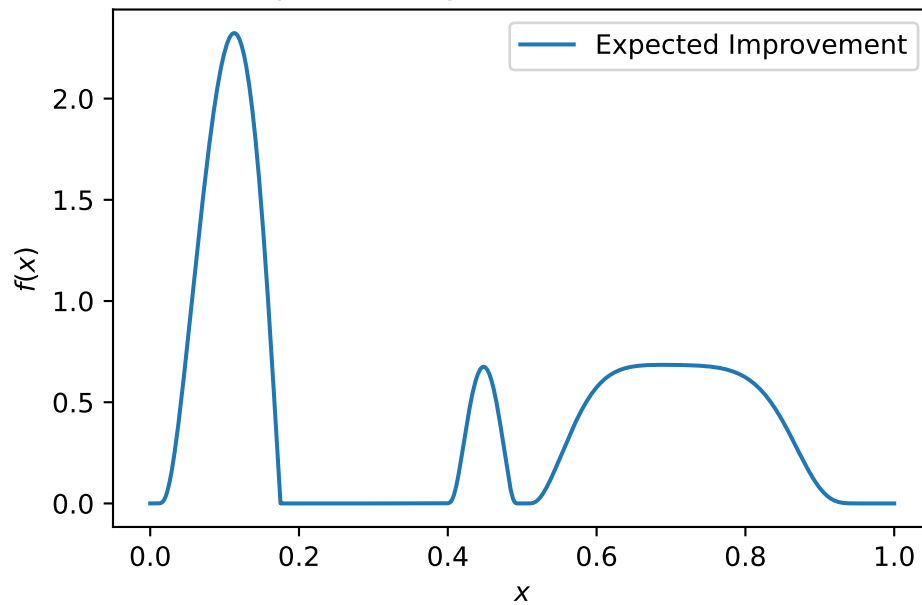
plt.scatter(X_train, y_train, label="Observations")
plt.plot(X, mean_prediction, label="Mean prediction")
if True:
    plt.fill_between(
        X.ravel(),
        mean_prediction - 2 * std_prediction,
        mean_prediction + 2 * std_prediction,
        alpha=0.5,
        label=r"95% confidence interval",
    )
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Gaussian process regression on noise-free dataset")

```



```
#plt.plot(X, y, label=r"$f(x) = x \sin(x)$", linestyle="dotted")
# plt.scatter(X_train, y_train, label="Observations")
plt.plot(X, -ei, label="Expected Improvement")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Gaussian process regression on noise-free dataset")
```

Gaussian process regression on noise-free dataset



7.11 Noise

```
import numpy as np
import spotPython
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from spotPython.design.spacefilling import spacefilling
from spotPython.build.kriging import Kriging
import matplotlib.pyplot as plt

gen = spacefilling(1)
rng = np.random.RandomState(1)
lower = np.array([-10])
upper = np.array([10])
fun = analytical().fun_sphere
fun_control = {"sigma": 2,
               "seed": 125}
X = gen.scipy_lhd(10, lower=lower, upper = upper)
print(X)
y = fun(X, fun_control=fun_control)
```

```

print(y)
y.shape
X_train = X.reshape(-1,1)
y_train = y

S = Kriging(name='kriging',
            seed=123,
            log_level=50,
            n_theta=1,
            noise=False)
S.fit(X_train, y_train)

X_axis = np.linspace(start=-13, stop=13, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S.predict(X_axis, return_val="all")

#plt.plot(X, y, label=r"$f(x) = x \sin(x)$", linestyle="dotted")
plt.scatter(X_train, y_train, label="Observations")
#plt.plot(X, ei, label="Expected Improvement")
plt.plot(X_axis, mean_prediction, label="mue")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Sphere: Gaussian process regression on noisy dataset")

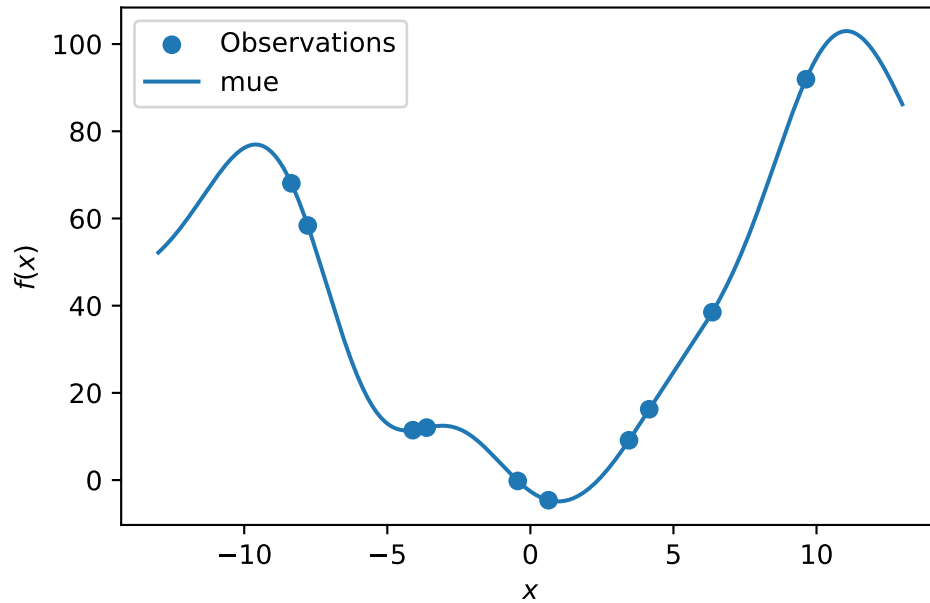
```

```

[[ 0.63529627]
 [-4.10764204]
 [-0.44071975]
 [ 9.63125638]
 [-8.3518118 ]
 [-3.62418901]
 [ 4.15331   ]
 [ 3.4468512 ]
 [ 6.36049088]
 [-7.77978539]]
[-4.61635371 11.44873209 -0.19988024 91.92791676 68.05926244 12.02926818
 16.2470957   9.12729929 38.4987029  58.38469104]

```

Sphere: Gaussian process regression on noisy dataset



S.log

```
{'negLnLike': array([24.69806131]),
 'theta': array([1.31023969]),
 'p': array([2.]),
 'Lambda': array([None], dtype=object)}
```

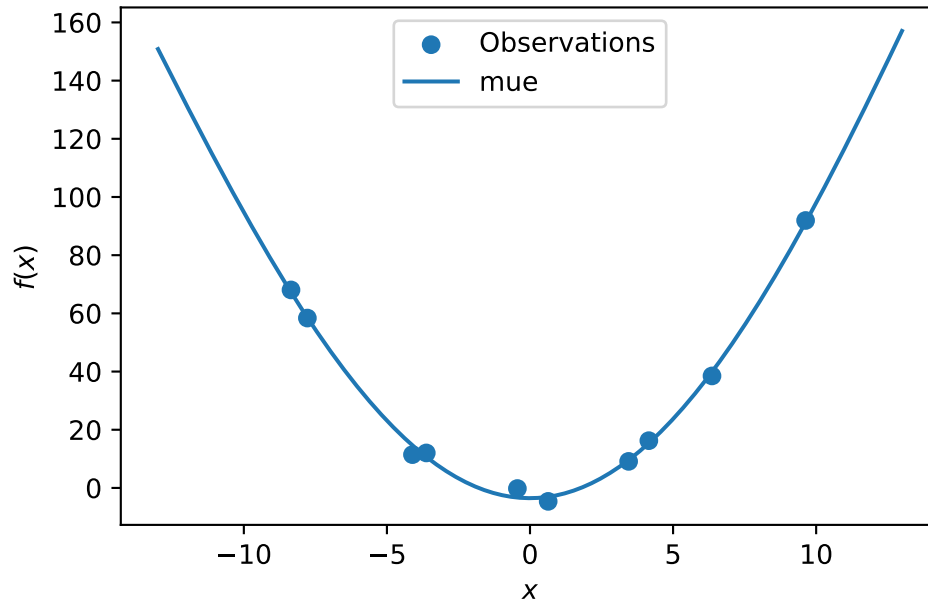
```
S = Kriging(name='kriging',
            seed=123,
            log_level=50,
            n_theta=1,
            noise=True)
S.fit(X_train, y_train)
```

```
X_axis = np.linspace(start=-13, stop=13, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S.predict(X_axis, return_val="all")
```

```
#plt.plot(X, y, label=r"$f(x) = x \sin(x)$", linestyle="dotted")
plt.scatter(X_train, y_train, label="Observations")
#plt.plot(X, ei, label="Expected Improvement")
plt.plot(X_axis, mean_prediction, label="mue")
```

```
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Sphere: Gaussian process regression with nugget on noisy dataset")
```

Sphere: Gaussian process regression with nugget on noisy dataset



S.log

```
{'negLnLike': array([22.14095646]),
 'theta': array([-0.32527397]),
 'p': array([2.]),
 'Lambda': array([9.08815016e-05])}
```

7.12 Cubic Function

```
import numpy as np
import spotPython
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from spotPython.design.spacefilling import spacefilling
```



```

from spotPython.build.kriging import Kriging
import matplotlib.pyplot as plt

gen = spacefilling(1)
rng = np.random.RandomState(1)
lower = np.array([-10])
upper = np.array([10])
fun = analytical().fun_cubed
fun_control = {"sigma": 10,
               "seed": 123}

X = gen.scipy_lhd(10, lower=lower, upper = upper)
print(X)
y = fun(X, fun_control=fun_control)
print(y)
y.shape
X_train = X.reshape(-1,1)
y_train = y

S = Kriging(name='kriging', seed=123, log_level=50, n_theta=1, noise=False)
S.fit(X_train, y_train)

X_axis = np.linspace(start=-13, stop=13, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S.predict(X_axis, return_val="all")

plt.scatter(X_train, y_train, label="Observations")
#plt.plot(X, ei, label="Expected Improvement")
plt.plot(X_axis, mean_prediction, label="mue")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Cubed: Gaussian process regression on noisy dataset")

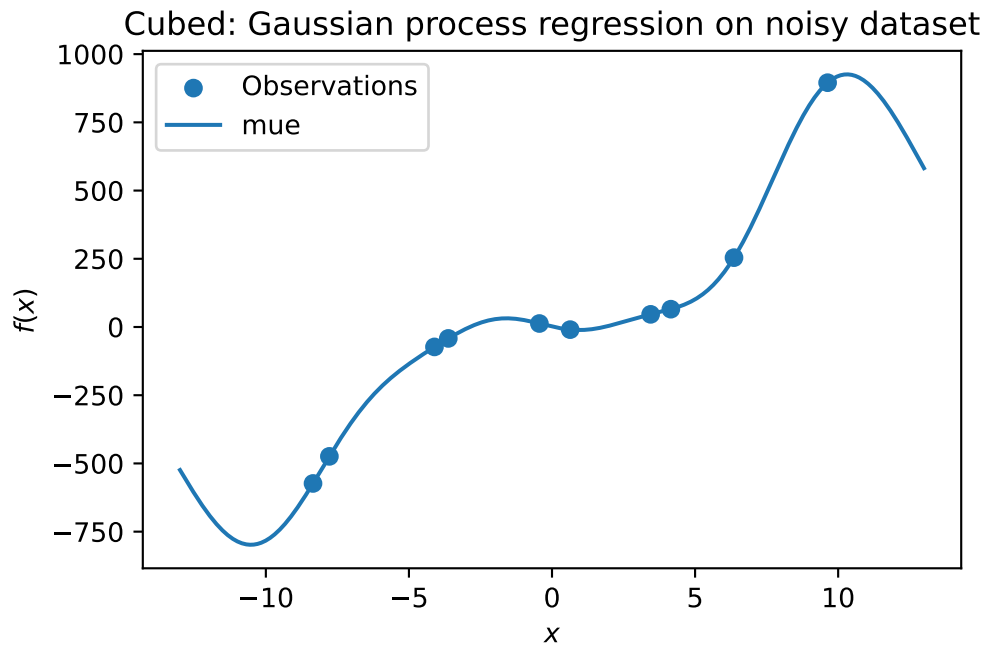
```

```

[[ 0.63529627]
 [-4.10764204]
 [-0.44071975]
 [ 9.63125638]
 [-8.3518118 ]
 [-3.62418901]
 [ 4.15331   ]
 [ 3.4468512 ]
 [ 6.36049088]

```

```
[-7.77978539]]
[ -9.63480707 -72.98497325  12.7936499   895.34567477 -573.35961837
 -41.83176425   65.27989461   46.37081417  254.1530734  -474.09587355]
```

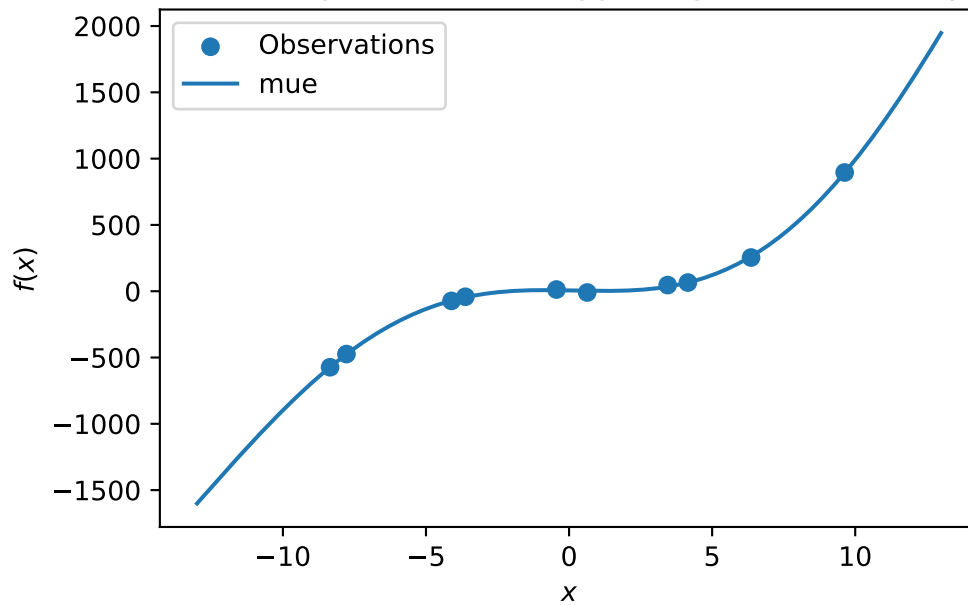


```
S = Kriging(name='kriging', seed=123, log_level=0, n_theta=1, noise=True)
S.fit(X_train, y_train)

X_axis = np.linspace(start=-13, stop=13, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S.predict(X_axis, return_val="all")

plt.scatter(X_train, y_train, label="Observations")
#plt.plot(X, ei, label="Expected Improvement")
plt.plot(X_axis, mean_prediction, label="mue")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Cubed: Gaussian process with nugget regression on noisy dataset")
```

Cubed: Gaussian process with nugget regression on noisy dataset



```
import numpy as np
import spotPython
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from spotPython.design.spacefilling import spacefilling
from spotPython.build.kriging import Kriging
import matplotlib.pyplot as plt

gen = spacefilling(1)
rng = np.random.RandomState(1)
lower = np.array([-10])
upper = np.array([10])
fun = analytical().fun_runge
fun_control = {"sigma": 0.25,
               "seed": 123}

X = gen.scipy_lhd(10, lower=lower, upper = upper)
print(X)
y = fun(X, fun_control=fun_control)
print(y)
y.shape
```

```

X_train = X.reshape(-1,1)
y_train = y

S = Kriging(name='kriging', seed=123, log_level=50, n_theta=1, noise=False)
S.fit(X_train, y_train)

X_axis = np.linspace(start=-13, stop=13, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S.predict(X_axis, return_val="all")

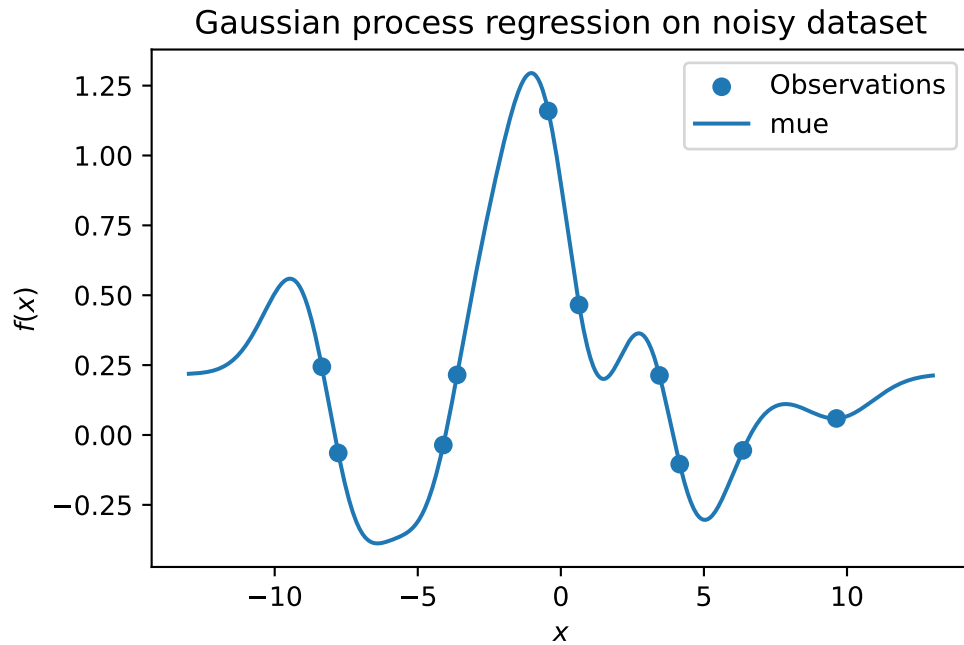
plt.scatter(X_train, y_train, label="Observations")
#plt.plot(X, ei, label="Expected Improvement")
plt.plot(X_axis, mean_prediction, label="mue")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Gaussian process regression on noisy dataset")

```

```

[[ 0.63529627]
 [-4.10764204]
 [-0.44071975]
 [ 9.63125638]
 [-8.3518118 ]
 [-3.62418901]
 [ 4.15331    ]
 [ 3.4468512 ]
 [ 6.36049088]
 [-7.77978539]]
[ 0.46517267 -0.03599548  1.15933822  0.05915901  0.24419145  0.21502359
 -0.10432134  0.21312309 -0.05502681 -0.06434374]

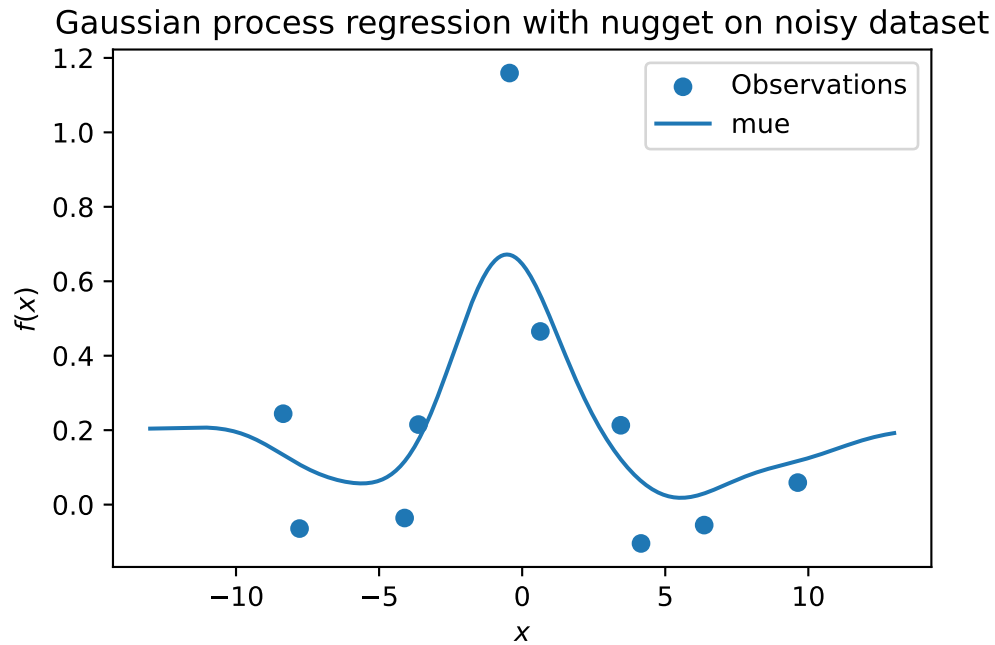
```



```
S = Kriging(name='kriging',
            seed=123,
            log_level=50,
            n_theta=1,
            noise=True)
S.fit(X_train, y_train)

X_axis = np.linspace(start=-13, stop=13, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S.predict(X_axis, return_val="all")

plt.scatter(X_train, y_train, label="Observations")
#plt.plot(X, ei, label="Expected Improvement")
plt.plot(X_axis, mean_prediction, label="mue")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Gaussian process regression with nugget on noisy dataset")
```



7.13 Factors

```
["num"] * 3
```

```
['num', 'num', 'num']
```

```
from spotPython.design.spacefilling import spacefilling
from spotPython.build.kriging import Kriging
from spotPython.fun.objectivefunctions import analytical
import numpy as np
```

```
gen = spacefilling(2)
n = 30
rng = np.random.RandomState(1)
lower = np.array([-5,-0])
upper = np.array([10,15])
fun = analytical().fun_branin_factor
#fun = analytical(sigma=0).fun_sphere
```

```

X0 = gen.scipy_lhd(n, lower=lower, upper = upper)
X1 = np.random.randint(low=1, high=3, size=(n,))
X = np.c_[X0, X1]
y = fun(X)
S = Kriging(name='kriging', seed=123, log_level=50, n_theta=3, noise=False, var_type=["nu
S.fit(X, y)
Sf = Kriging(name='kriging', seed=123, log_level=50, n_theta=3, noise=False, var_type=["n
Sf.fit(X, y)
n = 50
X0 = gen.scipy_lhd(n, lower=lower, upper = upper)
X1 = np.random.randint(low=1, high=3, size=(n,))
X = np.c_[X0, X1]
y = fun(X)
s=np.sum(np.abs(S.predict(X)[0] - y))
sf=np.sum(np.abs(Sf.predict(X)[0] - y))
sf - s

```

248.10101250325442

```
# vars(S)
```

```
# vars(Sf)
```

8 Hyperparameter Tuning and Noise

This chapter demonstrates how noisy functions can be handled by Spot.

8.1 Example: Spot and the Noisy Sphere Function

```
import numpy as np
from math import inf
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from scipy.optimize import shgo
from scipy.optimize import direct
from scipy.optimize import differential_evolution
import matplotlib.pyplot as plt
import os
import copy
import socket
from datetime import datetime
from dateutil.tz import tzlocal

start_time = datetime.now(tzlocal())
HOSTNAME = socket.gethostname().split(".")[0]
experiment_name = '10-sklearn' + "_" + HOSTNAME + "_" + str(start_time).split(".", 1)[0].r
experiment_name = experiment_name.replace(':', '-')
print(experiment_name)
if not os.path.exists('./figures'):
    os.makedirs('./figures')
```

10-sklearn_maans03_2023-07-03_10-13-29

8.1.1 The Objective Function: Noisy Sphere

- The spotPython package provides several classes of objective functions.

- We will use an analytical objective function with noise, i.e., a function that can be described by a (closed) formula:

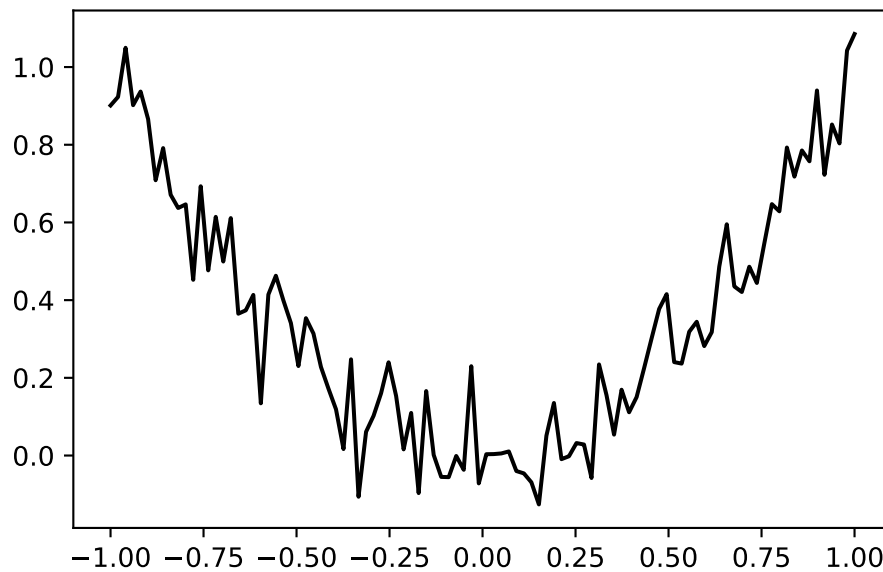
$$f(x) = x^2 + \epsilon$$

- Since `sigma` is set to 0.1, noise is added to the function:

```
fun = analytical().fun_sphere
fun_control = {"sigma": 0.1,
              "seed": 123}
```

- A plot illustrates the noise:

```
x = np.linspace(-1,1,100).reshape(-1,1)
y = fun(x, fun_control=fun_control)
plt.figure()
plt.plot(x,y, "k")
plt.show()
```



Spot is adopted as follows to cope with noisy functions:

1. `fun_repeats` is set to a value larger than 1 (here: 2)
2. `noise` is set to `true`. Therefore, a nugget (`Lambda`) term is added to the correlation matrix
3. `init size` (of the `design_control` dictionary) is set to a value larger than 1 (here: 2)

```

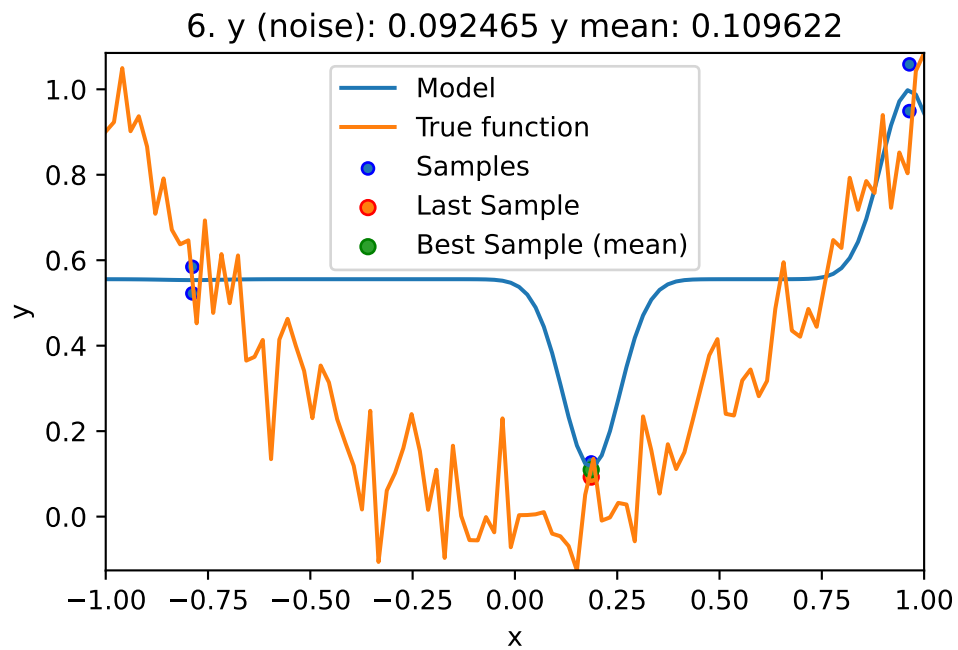
spot_1_noisy = spot.Spot(fun=fun,
    lower = np.array([-1]),
    upper = np.array([1]),
    fun_evals = 10,
    fun_repeats = 2,
    noise = True,
    seed=123,
    show_models=True,
    fun_control = fun_control,
    design_control={"init_size": 3,
        "repeats": 2},
    surrogate_control={"noise": True})

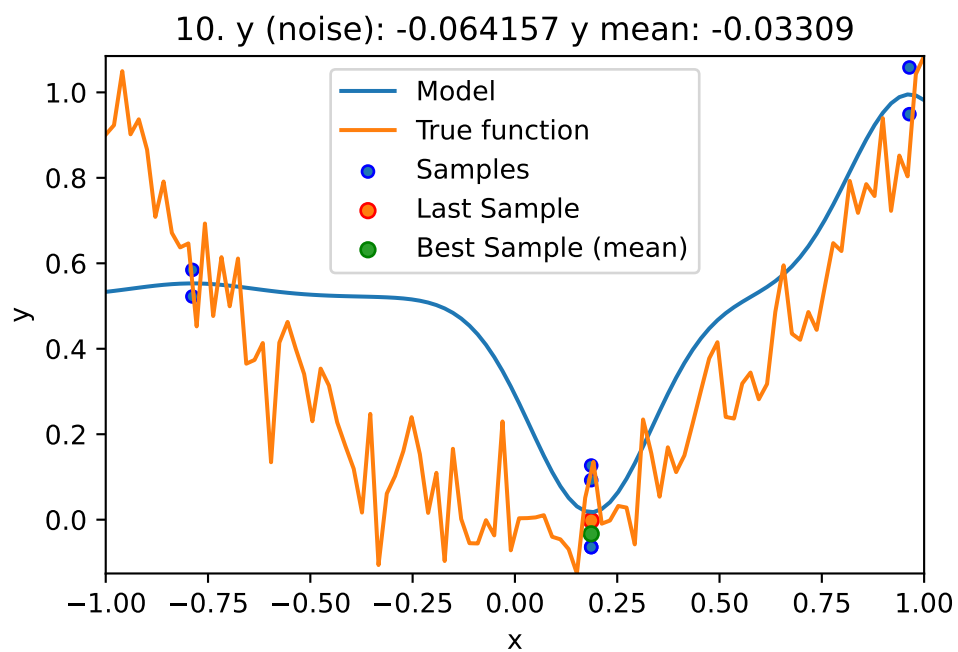
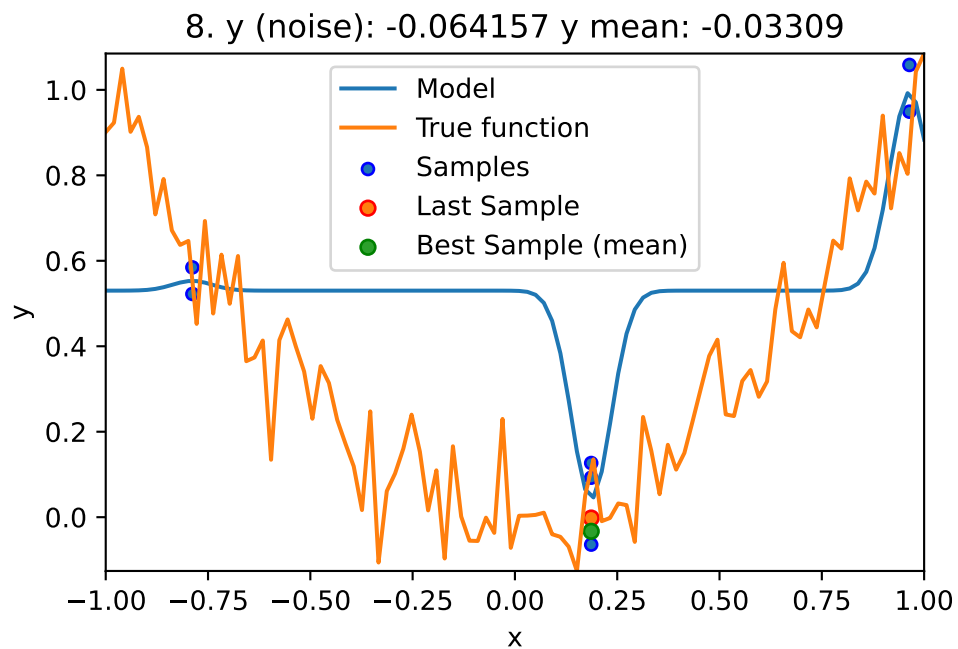
```

```

spot_1_noisy.run()

```





<spotPython.spot.spot.Spot at 0x1599b8d30>

8.2 Print the Results

```
spot_1_noisy.print_results()
```

```
min y: -0.06415721563564872
x0: 0.18642671321228718
min mean y: -0.03309048069165033
x0: 0.18642671321228718
```

```
[['x0', 0.18642671321228718], ['x0', 0.18642671321228718]]
```

```
spot_1_noisy.plot_progress(log_y=False,
                             filename="./figures/" + experiment_name+"_progress.png")
```

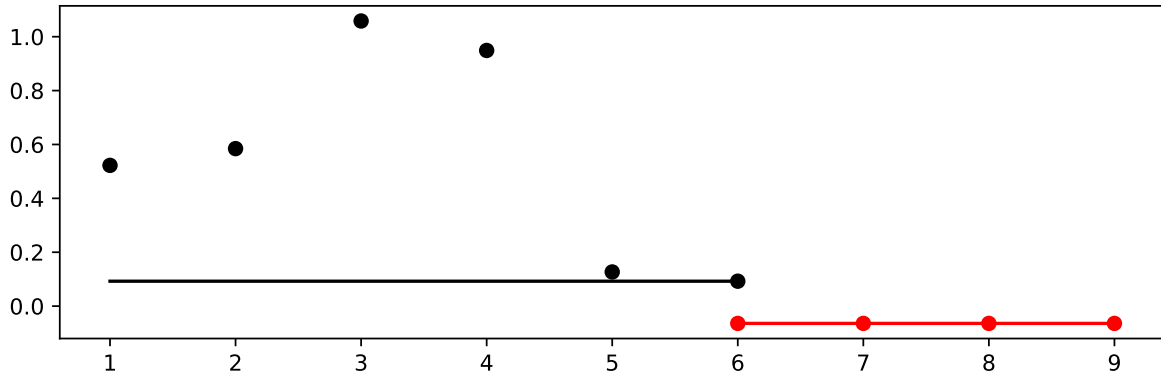


Figure 8.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

8.3 Noise and Surrogates: The Nugget Effect

8.3.1 The Noisy Sphere

8.3.1.1 The Data

- We prepare some data first:

```

import numpy as np
import spotPython
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from spotPython.design.spacefilling import spacefilling
from spotPython.build.kriging import Kriging
import matplotlib.pyplot as plt

gen = spacefilling(1)
rng = np.random.RandomState(1)
lower = np.array([-10])
upper = np.array([10])
fun = analytical().fun_sphere
fun_control = {"sigma": 2,
               "seed": 125}
X = gen.scipy_lhd(10, lower=lower, upper = upper)
y = fun(X, fun_control=fun_control)
X_train = X.reshape(-1,1)
y_train = y

```

- A surrogate without nugget is fitted to these data:

```

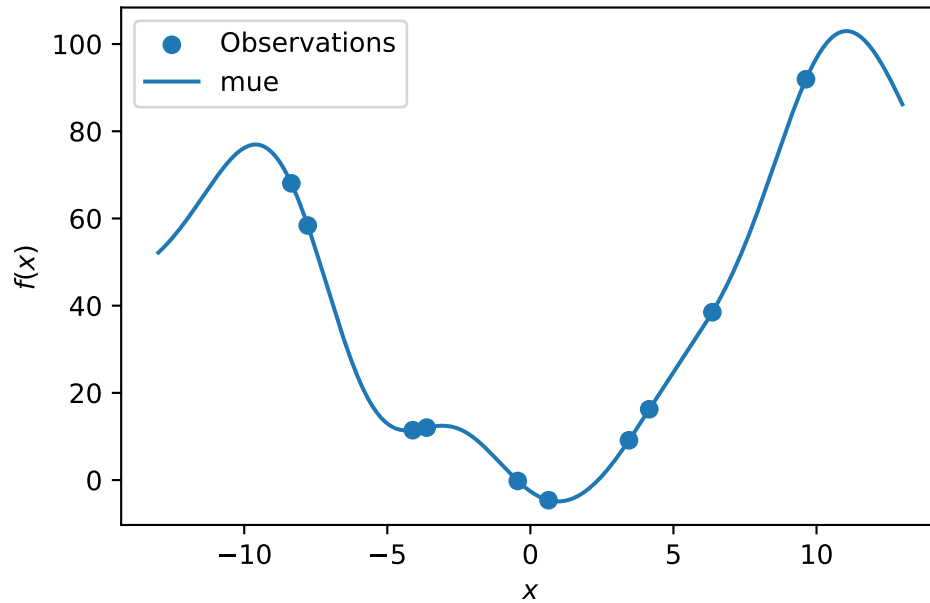
S = Kriging(name='kriging',
            seed=123,
            log_level=50,
            n_theta=1,
            noise=False)
S.fit(X_train, y_train)

X_axis = np.linspace(start=-13, stop=13, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S.predict(X_axis, return_val="all")

plt.scatter(X_train, y_train, label="Observations")
plt.plot(X_axis, mean_prediction, label="mue")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Sphere: Gaussian process regression on noisy dataset")

```

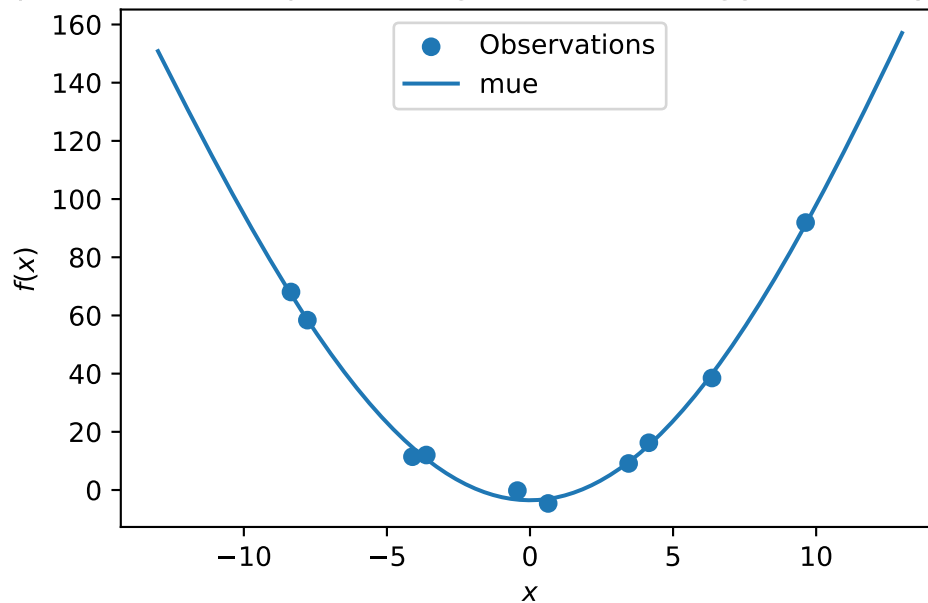
Sphere: Gaussian process regression on noisy dataset



- In comparison to the surrogate without nugget, we fit a surrogate with nugget to the data:

```
S_nug = Kriging(name='kriging',
                seed=123,
                log_level=50,
                n_theta=1,
                noise=True)
S_nug.fit(X_train, y_train)
X_axis = np.linspace(start=-13, stop=13, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S_nug.predict(X_axis, return_val="all")
plt.scatter(X_train, y_train, label="Observations")
plt.plot(X_axis, mean_prediction, label="mue")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Sphere: Gaussian process regression with nugget on noisy dataset")
```

Sphere: Gaussian process regression with nugget on noisy dataset



- The value of the nugget term can be extracted from the model as follows:

```
S.Lambda
```

```
S_nug.Lambda
```

```
9.088149957981389e-05
```

- We see:
 - the first model `S` has no nugget,
 - whereas the second model has a nugget value (`Lambda`) larger than zero.

8.4 Exercises

8.4.1 Noisy fun_cubed

- Analyse the effect of noise on the `fun_cubed` function with the following settings:

```
fun = analytical().fun_cubed  
fun_control = {"sigma": 10,
```

```
        "seed": 123}
lower = np.array([-10])
upper = np.array([10])
```

8.4.2 fun_runge

- Analyse the effect of noise on the `fun_runge` function with the following settings:

```
lower = np.array([-10])
upper = np.array([10])
fun = analytical().fun_runge
fun_control = {"sigma": 0.25,
               "seed": 123}
```

8.4.3 fun_forrester

- Analyse the effect of noise on the `fun_forrester` function with the following settings:

```
lower = np.array([0])
upper = np.array([1])
fun = analytical().fun_forrester
fun_control = {"sigma": 5,
               "seed": 123}
```

8.4.4 fun_xsin

- Analyse the effect of noise on the `fun_xsin` function with the following settings:

```
lower = np.array([-1.])
upper = np.array([1.])
fun = analytical().fun_xsin
fun_control = {"sigma": 0.5,
               "seed": 123}
```


9 Handling Noise: Optimal Computational Budget Allocation in Spot

This notebook demonstrates how noisy functions can be handled with OCBA by Spot.

9.1 Example: Spot, OCBA, and the Noisy Sphere Function

```
import numpy as np
from math import inf
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from scipy.optimize import shgo
from scipy.optimize import direct
from scipy.optimize import differential_evolution
import matplotlib.pyplot as plt
```

9.1.1 The Objective Function: Noisy Sphere

The `spotPython` package provides several classes of objective functions. We will use an analytical objective function with noise, i.e., a function that can be described by a (closed) formula:

$$f(x) = x^2 + \epsilon$$

Since `sigma` is set to 0.1, noise is added to the function:

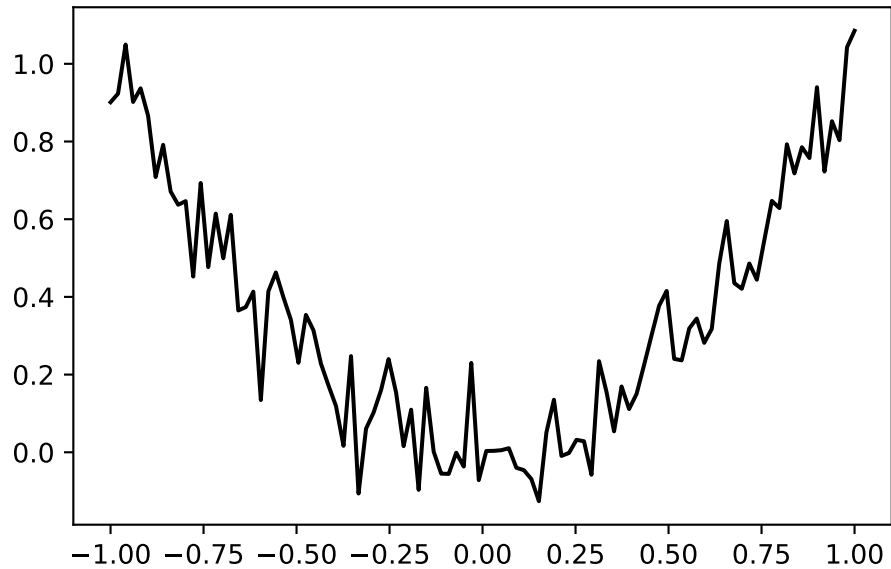
```
fun = analytical().fun_sphere
fun_control = {"sigma": 0.1,
              "seed": 123}
```

A plot illustrates the noise:

```

x = np.linspace(-1,1,100).reshape(-1,1)
y = fun(x, fun_control=fun_control)
plt.figure()
plt.plot(x,y, "k")
plt.show()

```



Spot is adopted as follows to cope with noisy functions:

1. `fun_repeats` is set to a value larger than 1 (here: 2)
2. `noise` is set to `true`. Therefore, a nugget (`Lambda`) term is added to the correlation matrix
3. `init size` (of the `design_control` dictionary) is set to a value larger than 1 (here: 2)

```

spot_1_noisy = spot.Spot(fun=fun,
    lower = np.array([-1]),
    upper = np.array([1]),
    fun_evals = 50,
    fun_repeats = 2,
    infill_criterion="ei",
    noise = True,
    tolerance_x=0.0,
    ocba_delta = 1,
    seed=123,

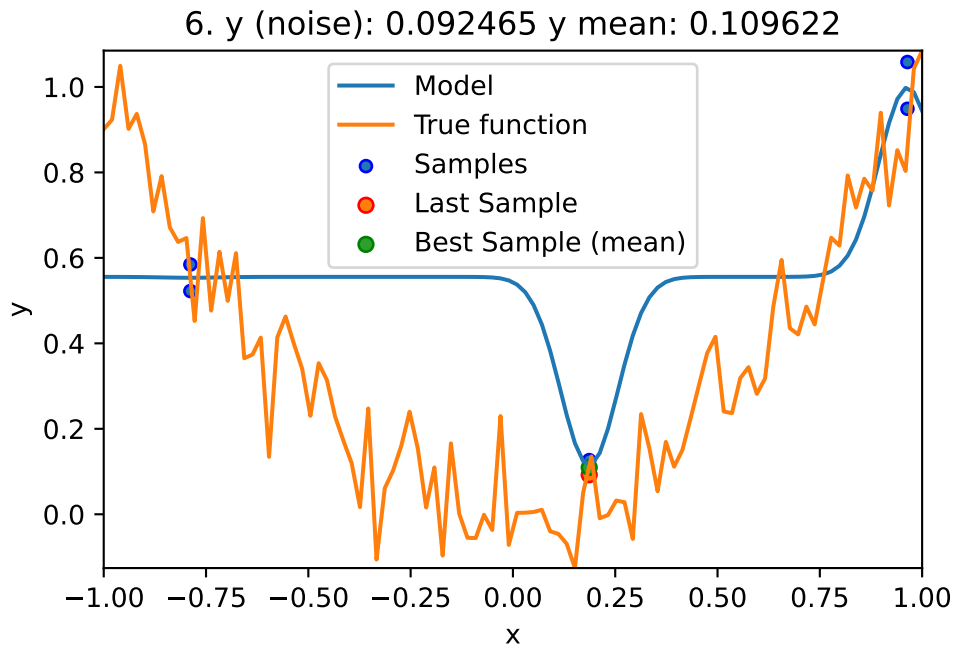
```

```

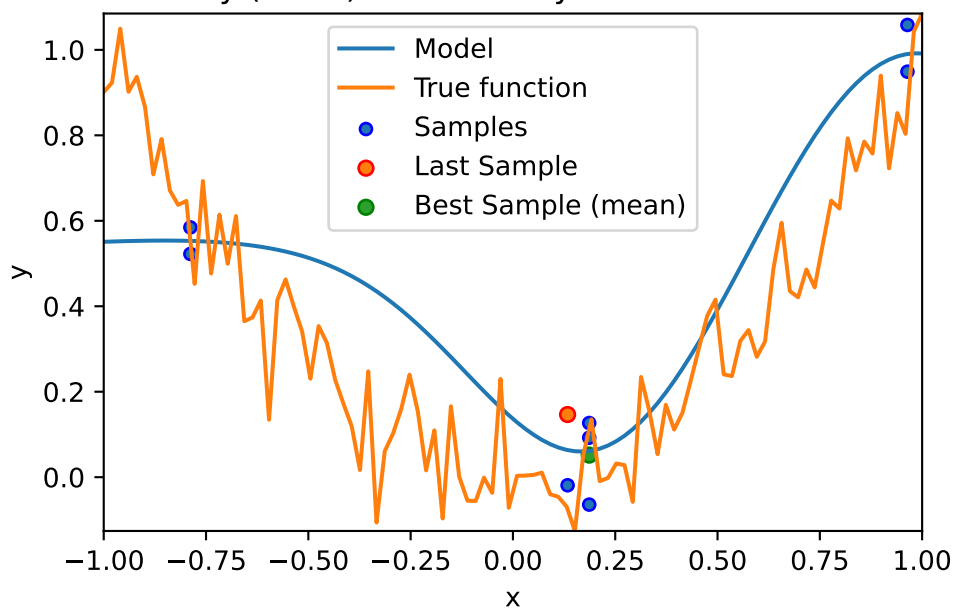
show_models=True,
fun_control = fun_control,
design_control={"init_size": 3,
               "repeats": 2},
surrogate_control={"noise": True})

```

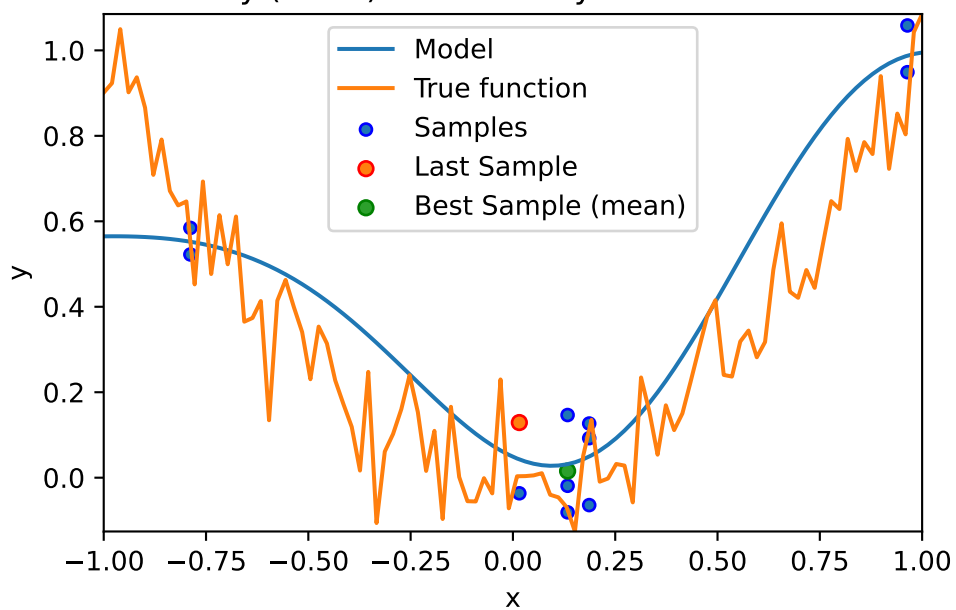
```
spot_1_noisy.run()
```

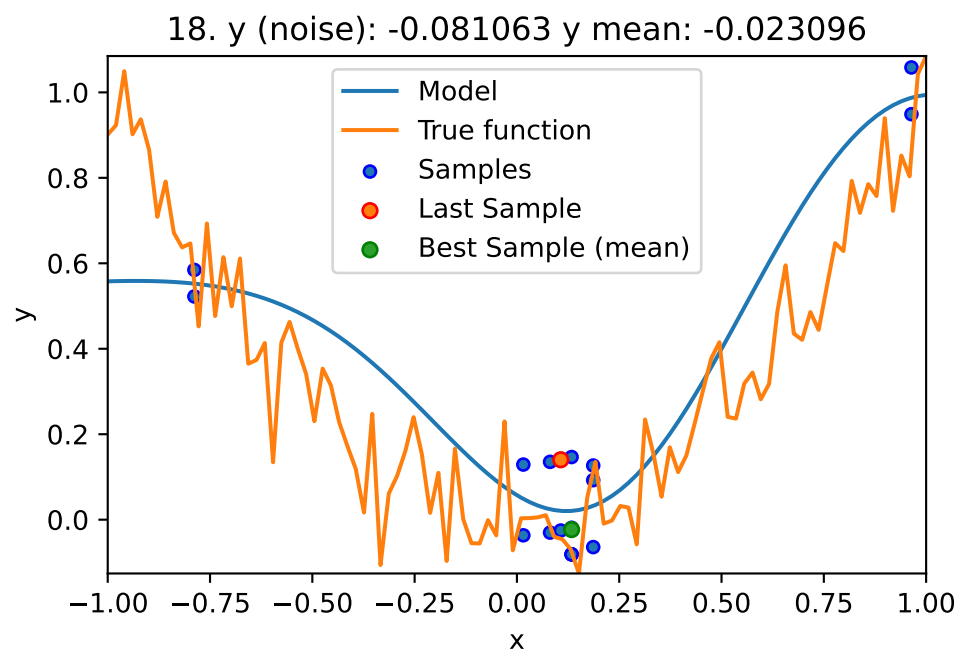
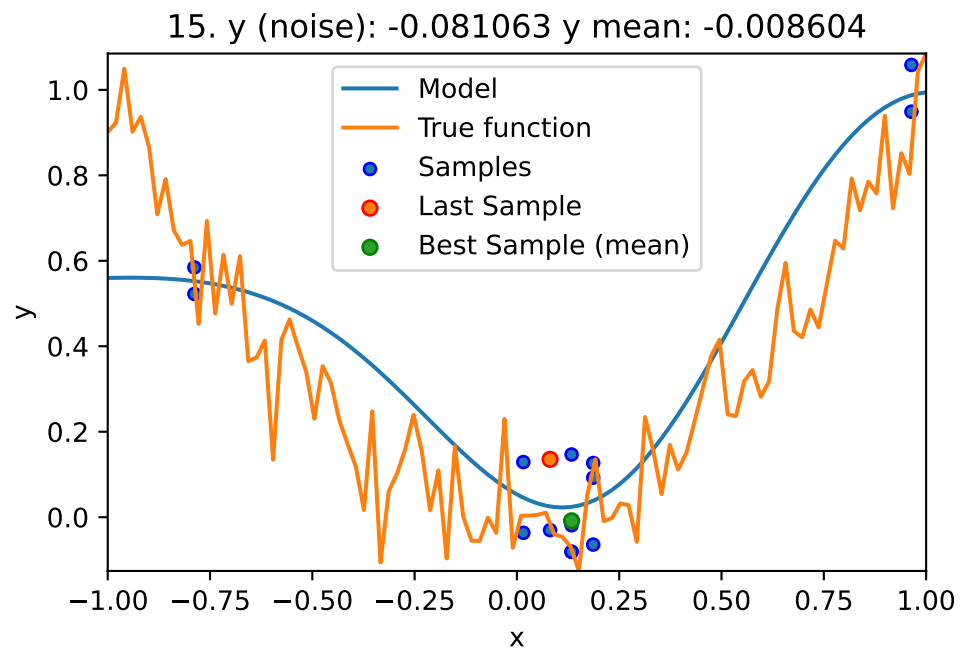


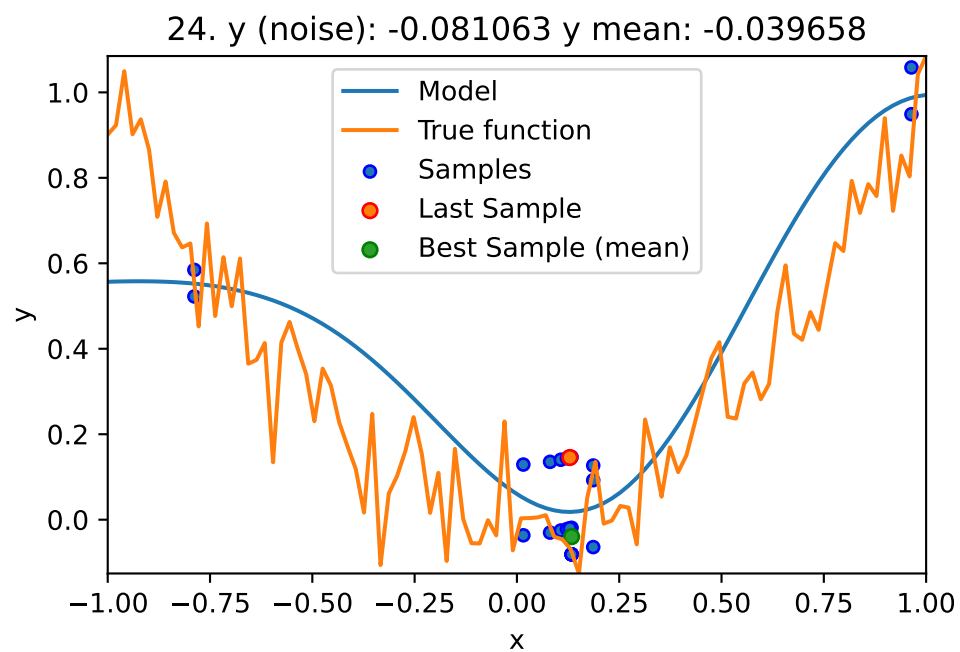
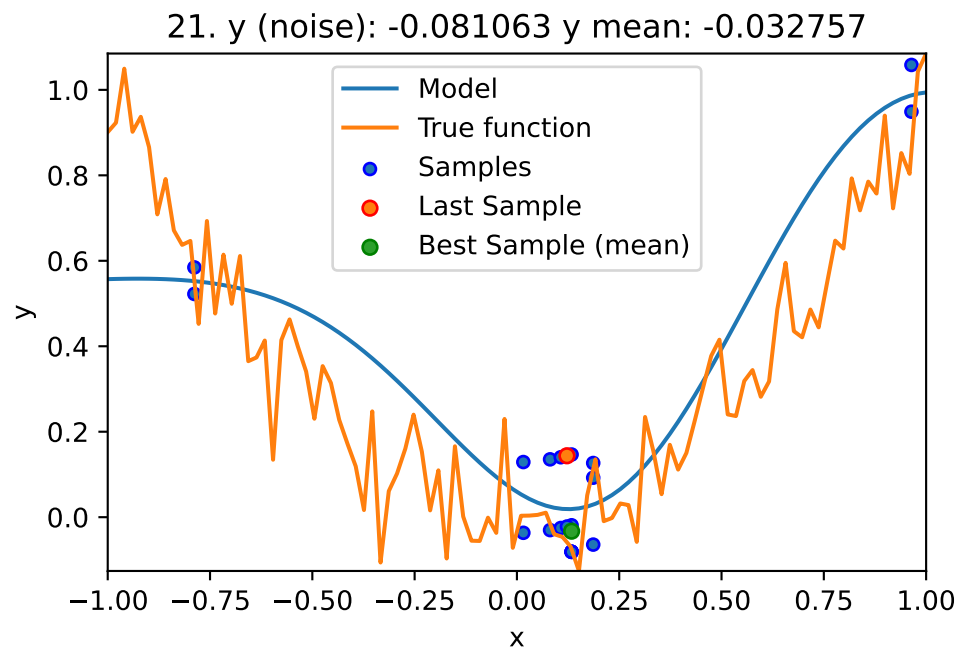
9. y (noise): -0.064157 y mean: 0.051695

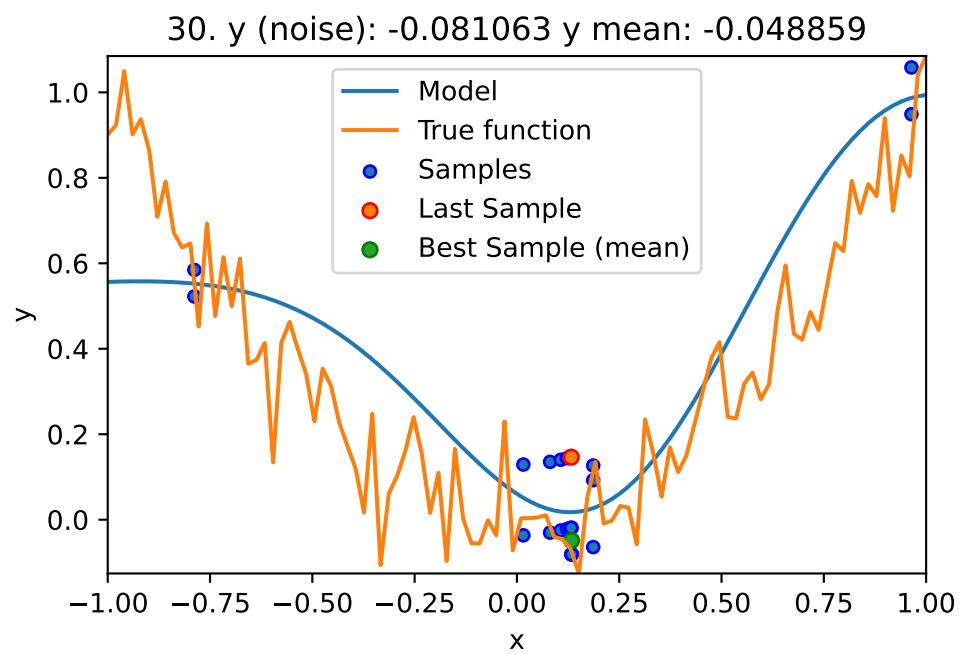
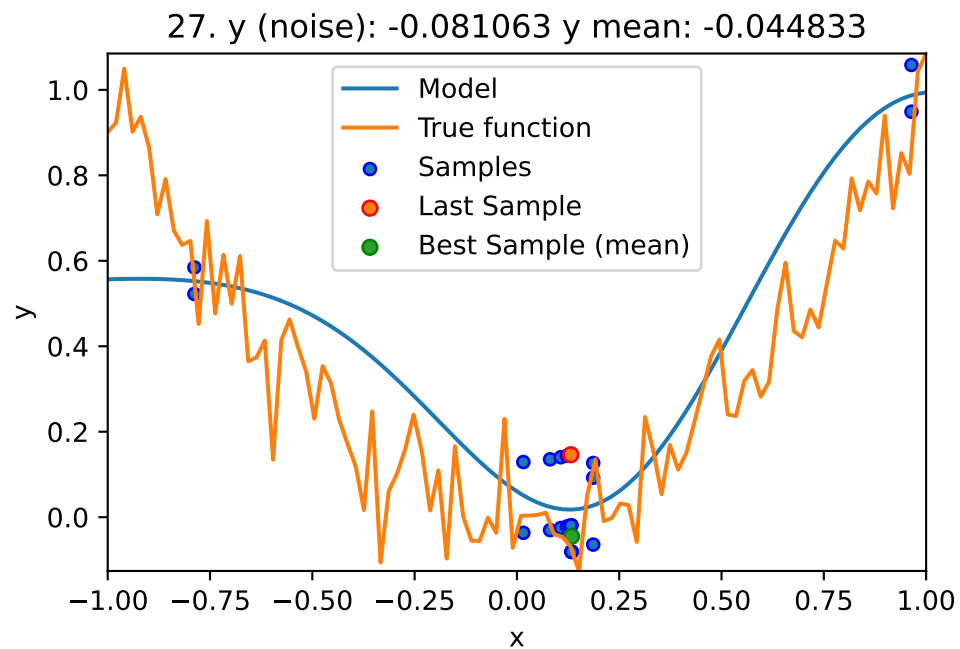


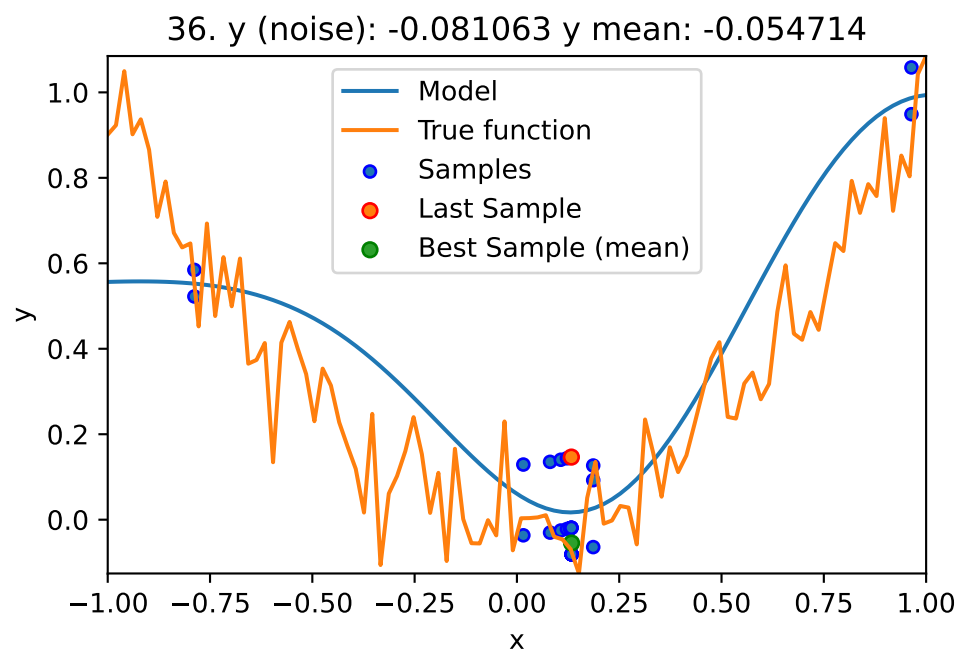
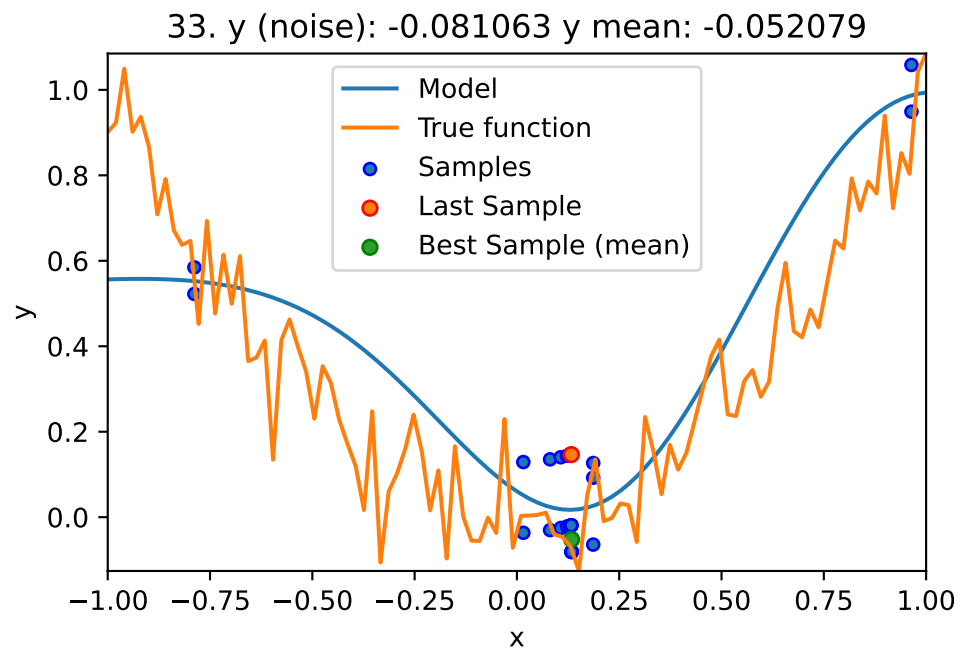
12. y (noise): -0.081063 y mean: 0.01555



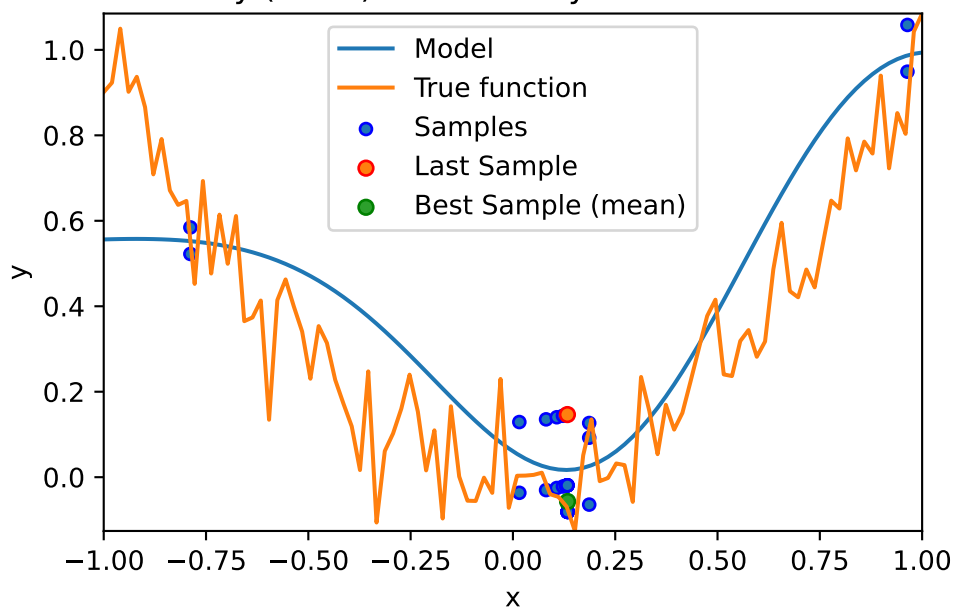




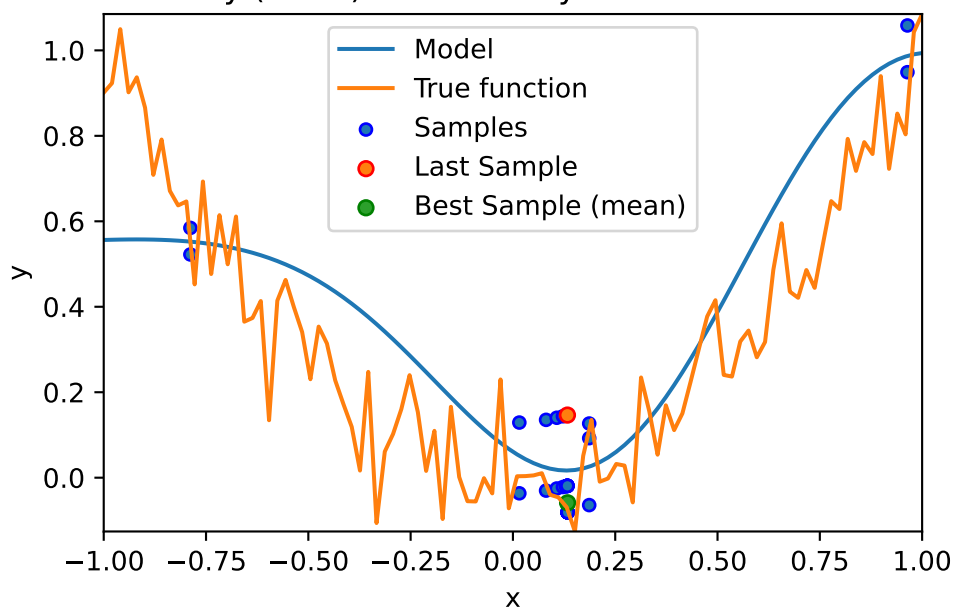




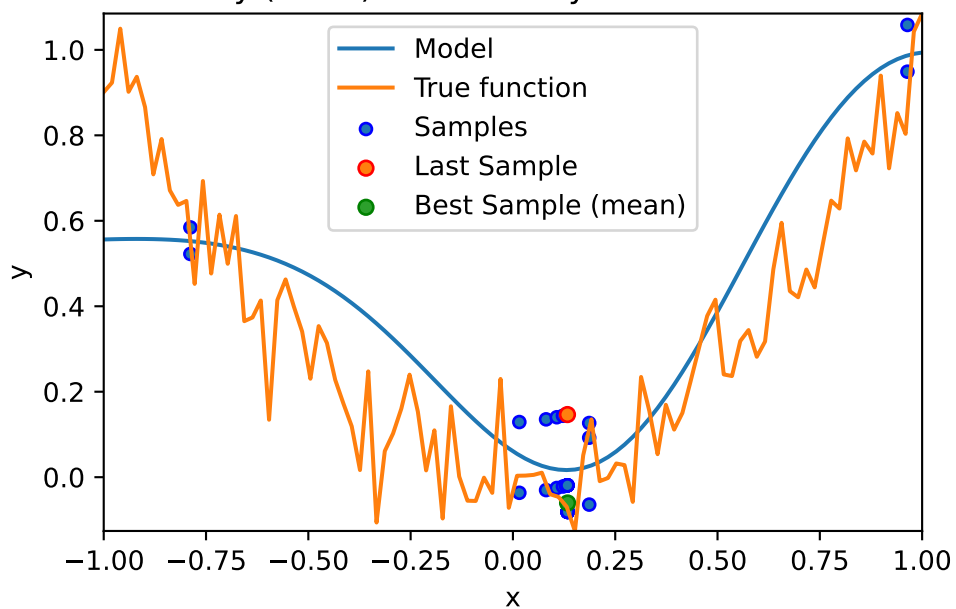
39. y (noise): -0.081063 y mean: -0.05691



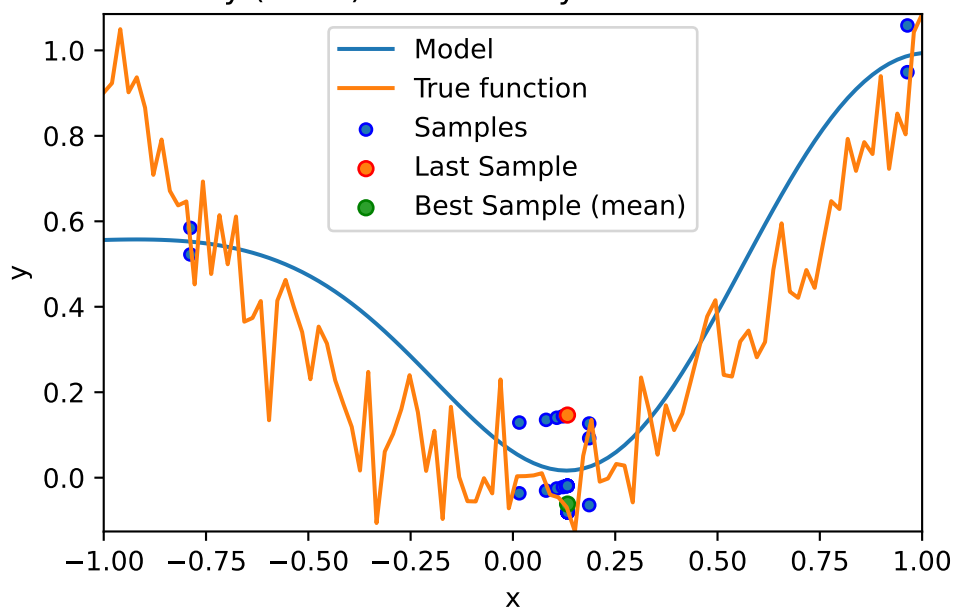
42. y (noise): -0.081063 y mean: -0.058768

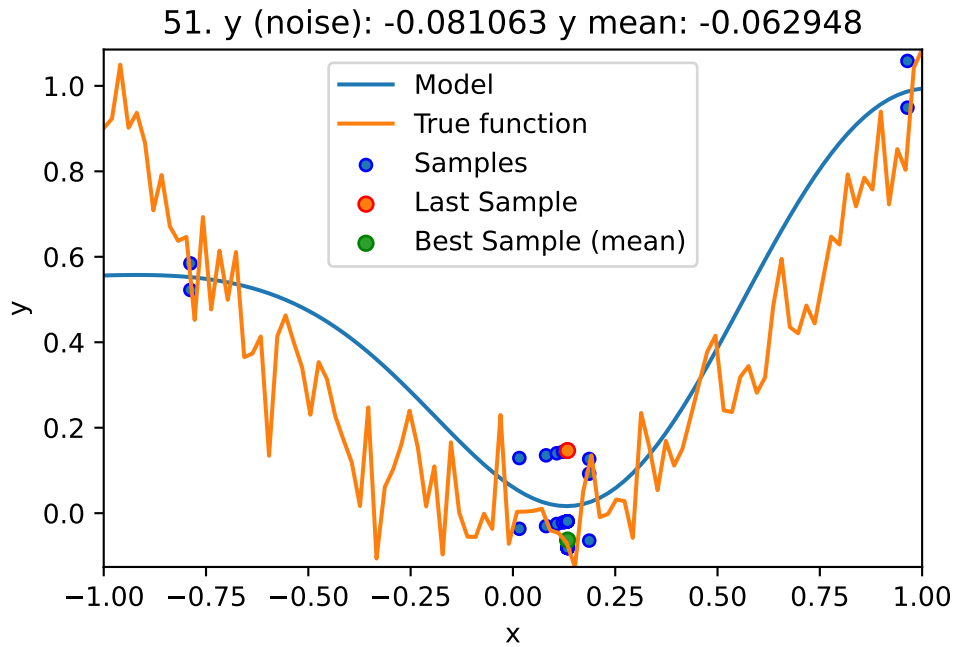


45. y (noise): -0.081063 y mean: -0.06036



48. y (noise): -0.081063 y mean: -0.061741





```
<spotPython.spot.spot.Spot at 0x15fa25360>
```

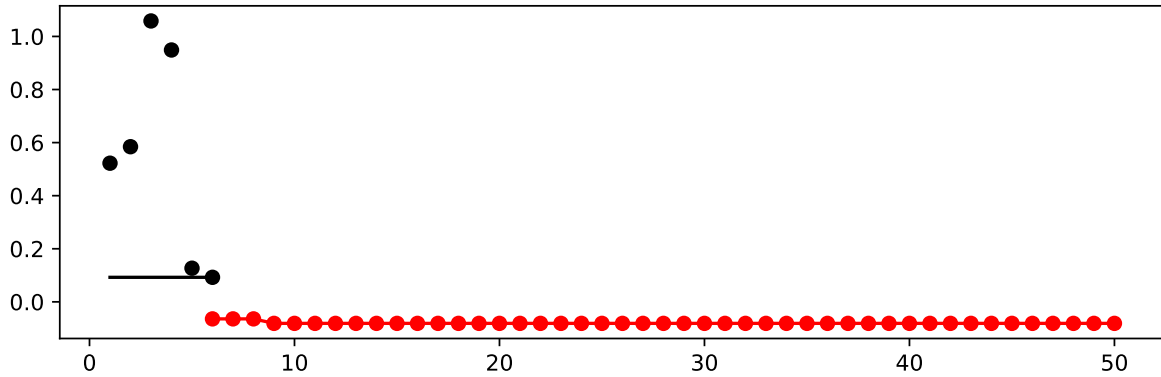
9.2 Print the Results

```
spot_1_noisy.print_results()
```

```
min y: -0.08106318976988831
x0: 0.13359994485364424
min mean y: -0.06294830657915665
x0: 0.13359994485364424
```

```
[['x0', 0.13359994485364424], ['x0', 0.13359994485364424]]
```

```
spot_1_noisy.plot_progress(log_y=False)
```



9.3 Noise and Surrogates: The Nugget Effect

9.3.1 The Noisy Sphere

9.3.1.1 The Data

We prepare some data first:

```
import numpy as np
import spotPython
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from spotPython.design.spacefilling import spacefilling
from spotPython.build.kriging import Kriging
import matplotlib.pyplot as plt

gen = spacefilling(1)
rng = np.random.RandomState(1)
lower = np.array([-10])
upper = np.array([10])
fun = analytical().fun_sphere
fun_control = {"sigma": 2,
               "seed": 125}
X = gen.scipy_lhd(10, lower=lower, upper = upper)
y = fun(X, fun_control=fun_control)
X_train = X.reshape(-1,1)
y_train = y
```

A surrogate without nugget is fitted to these data:

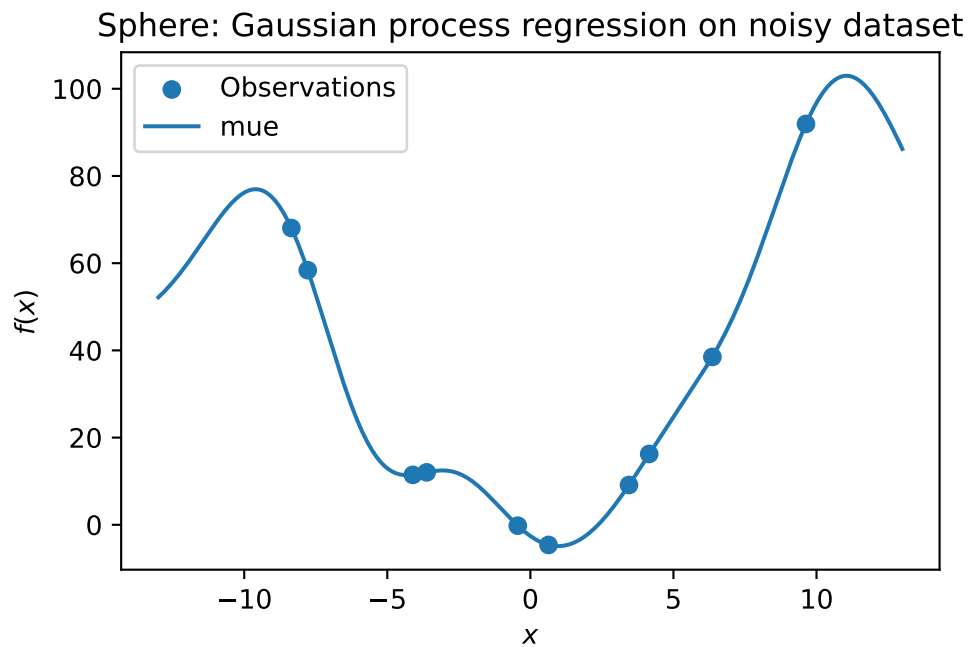
```

S = Kriging(name='kriging',
            seed=123,
            log_level=50,
            n_theta=1,
            noise=False)
S.fit(X_train, y_train)

X_axis = np.linspace(start=-13, stop=13, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S.predict(X_axis, return_val="all")

plt.scatter(X_train, y_train, label="Observations")
plt.plot(X_axis, mean_prediction, label="mue")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Sphere: Gaussian process regression on noisy dataset")

```



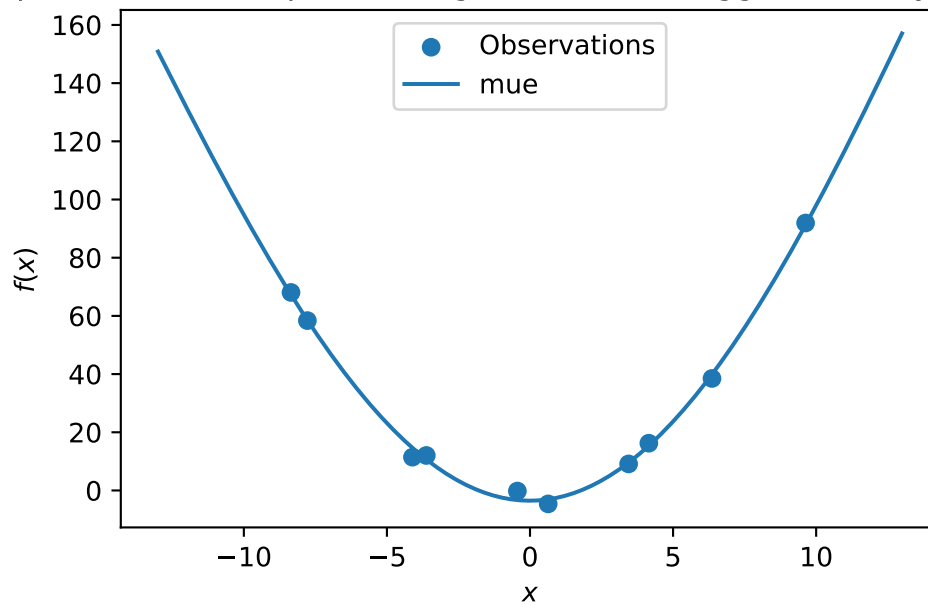
In comparison to the surrogate without nugget, we fit a surrogate with nugget to the data:

```

S_nug = Kriging(name='kriging',
                seed=123,
                log_level=50,
                n_theta=1,
                noise=True)
S_nug.fit(X_train, y_train)
X_axis = np.linspace(start=-13, stop=13, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S_nug.predict(X_axis, return_val="all")
plt.scatter(X_train, y_train, label="Observations")
plt.plot(X_axis, mean_prediction, label="mue")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Sphere: Gaussian process regression with nugget on noisy dataset")

```

Sphere: Gaussian process regression with nugget on noisy dataset



The value of the nugget term can be extracted from the model as follows:

```
S.Lambda
```

```
S_nug.Lambda
```

9.088149972334418e-05

We see:

- the first model S has no nugget,
- whereas the second model has a nugget value (Lambda) larger than zero.

9.4 Exercises

9.4.1 Noisy fun_cubed

Analyse the effect of noise on the `fun_cubed` function with the following settings:

```
fun = analytical().fun_cubed
fun_control = {"sigma": 10,
              "seed": 123}
lower = np.array([-10])
upper = np.array([10])
```

9.4.2 fun_runge

Analyse the effect of noise on the `fun_runge` function with the following settings:

```
lower = np.array([-10])
upper = np.array([10])
fun = analytical().fun_runge
fun_control = {"sigma": 0.25,
              "seed": 123}
```

9.4.3 fun_forrester

Analyse the effect of noise on the `fun_forrester` function with the following settings:

```
lower = np.array([0])
upper = np.array([1])
fun = analytical().fun_forrester
fun_control = {"sigma": 5,
              "seed": 123}
```

9.4.4 fun_xsin

Analyse the effect of noise on the `fun_xsin` function with the following settings:

```
lower = np.array([-1.])
upper = np.array([1.])
fun = analytical().fun_xsin
fun_control = {"sigma": 0.5,
               "seed": 123}

spot_1_noisy.mean_y.shape[0]
```


10 HPT: sklearn SVC on Moons Data

This document refers to the following software versions:

- python: 3.10.10

```
pip list | grep "spot[RiverPython]"
```

spotPython	0.2.52
spotRiver	0.0.94

Note: you may need to restart the kernel to use updated packages.

spotPython can be installed via pip. Alternatively, the source code can be downloaded from gitHub: <https://github.com/sequential-parameter-optimization/spotPython>.

```
!pip install spotPython
```

- Uncomment the following lines if you want to for (re-)installation the latest version of spotPython from gitHub.

```
# import sys
# !{sys.executable} -m pip install --upgrade build
# !{sys.executable} -m pip install --upgrade --force-reinstall spotPython
```

10.1 Step 1: Setup

Before we consider the detailed experimental setup, we select the parameters that affect run time and the initial design size.

```
MAX_TIME = 1
INIT_SIZE = 5
```

```

import os
import copy
import socket
from datetime import datetime
from dateutil.tz import tzlocal
start_time = datetime.now(tzlocal())
HOSTNAME = socket.gethostname().split(".")[0]
experiment_name = '10-sklearn' + "_" + HOSTNAME + "_" + str(MAX_TIME) + "min_" + str(INIT_
experiment_name = experiment_name.replace(':', '-')
print(experiment_name)
if not os.path.exists('./figures'):
    os.makedirs('./figures')

```

10-sklearn_maans03_1min_5init_2023-07-03_10-15-36

10.2 Step 2: Initialization of the Empty fun_control Dictionary

 Caution: Tensorboard does not work under Windows

- Since tensorboard does not work under Windows, we recommend setting the parameter `tensorboard_path` to `None` if you are working under Windows.

```

from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="classification",
    tensorboard_path="runs/10_spot_hpt_sklearn_classification")

```

10.3 Step 3: SKlearn Load Data (Classification)

Randomly generate classification data.

```

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons, make_circles, make_classification
n_features = 2
n_samples = 250
target_column = "y"

```

```

ds = make_moons(n_samples, noise=0.5, random_state=0)
X, y = ds
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.4, random_state=42
)
train = pd.DataFrame(np.hstack((X_train, y_train.reshape(-1, 1))))
test = pd.DataFrame(np.hstack((X_test, y_test.reshape(-1, 1))))
train.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
test.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
train.head()

```

	x1	x2	y
0	1.083978	-1.246111	1.0
1	0.074916	0.868104	0.0
2	-1.668535	0.751752	0.0
3	1.286597	1.454165	0.0
4	1.387021	0.448355	1.0

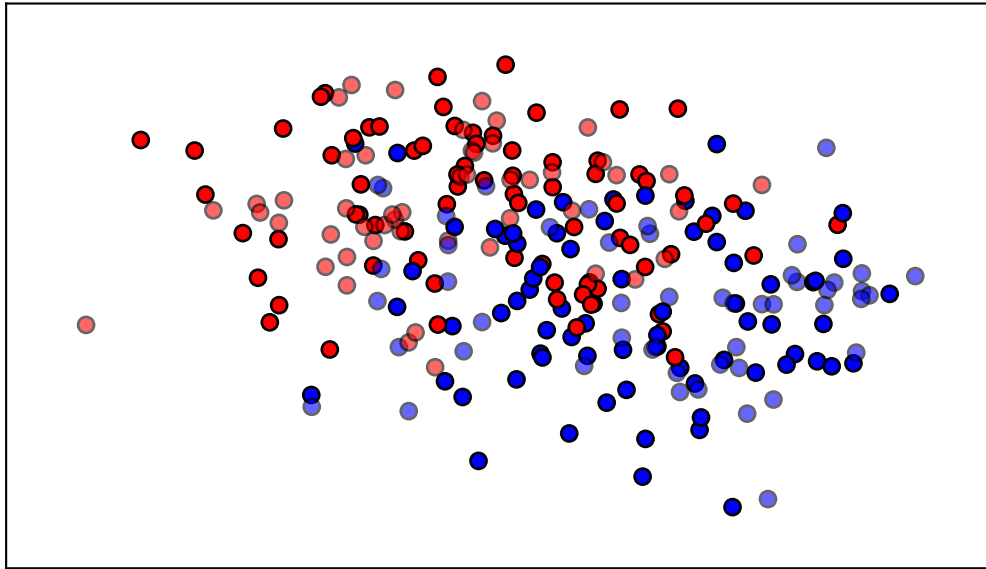
```

import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
cm = plt.cm.RdBu
cm_bright = ListedColormap(["#FF0000", "#0000FF"])
ax = plt.subplot(1, 1, 1)
ax.set_title("Input data")
# Plot the training points
ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright, edgecolors="k")
# Plot the testing points
ax.scatter(
    X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright, alpha=0.6, edgecolors="k"
)
ax.set_xlim(x_min, x_max)
ax.set_ylim(y_min, y_max)
ax.set_xticks(())
ax.set_yticks(())
plt.tight_layout()
plt.show()

```

Input data



```
n_samples = len(train)
# add the dataset to the fun_control
fun_control.update({"data": None, # dataset,
                  "train": train,
                  "test": test,
                  "n_samples": n_samples,
                  "target_column": target_column})
```

10.4 Step 4: Specification of the Preprocessing Model

Data preprocessing can be very simple, e.g., you can ignore it. Then you would choose the `prep_model` "None":

```
prep_model = None
fun_control.update({"prep_model": prep_model})
```

A default approach for numerical data is the `StandardScaler` (mean 0, variance 1). This can be selected as follows:

```

from sklearn.preprocessing import StandardScaler
prep_model = StandardScaler()
fun_control.update({"prep_model": prep_model})

```

Even more complicated pre-processing steps are possible, e.g., the following pipeline:

```

# categorical_columns = []
# one_hot_encoder = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
# prep_model = ColumnTransformer(
#     transformers=[
#         ("categorical", one_hot_encoder, categorical_columns),
#     ],
#     remainder=StandardScaler(),
# )

```

10.5 Step 5: Select Model (algorithm) and core_model_hyper_dict

The selection of the algorithm (ML model) that should be tuned is done by specifying the its name from the `sklearn` implementation. For example, the SVC support vector machine classifier is selected as follows:

```

from sklearn.linear_model import RidgeCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import ElasticNet
from spotPython.hyperparameters.values import add_core_model_to_fun_control
from spotPython.data.sklearn_hyper_dict import SklearnHyperDict
from spotPython.fun.hypersklearn import HyperSklearn

# core_model = RidgeCV
# core_model = GradientBoostingRegressor
# core_model = ElasticNet
# core_model = RandomForestClassifier
core_model = SVC

```

```

# core_model = LogisticRegression
# core_model = KNeighborsClassifier
# core_model = GradientBoostingClassifier
fun_control = add_core_model_to_fun_control(core_model=core_model,
                                           fun_control=fun_control,
                                           hyper_dict=SklearnHyperDict,
                                           filename=None)

```

Now `fun_control` has the information from the JSON file. The corresponding entries for the `core_model` class are shown below.

```
fun_control['core_model_hyper_dict']
```

```

{'C': {'type': 'float',
      'default': 1.0,
      'transform': 'None',
      'lower': 0.1,
      'upper': 10.0},
 'kernel': {'levels': ['linear', 'poly', 'rbf', 'sigmoid'],
            'type': 'factor',
            'default': 'rbf',
            'transform': 'None',
            'core_model_parameter_type': 'str',
            'lower': 0,
            'upper': 3},
 'degree': {'type': 'int',
            'default': 3,
            'transform': 'None',
            'lower': 3,
            'upper': 3},
 'gamma': {'levels': ['scale', 'auto'],
          'type': 'factor',
          'default': 'scale',
          'transform': 'None',
          'core_model_parameter_type': 'str',
          'lower': 0,
          'upper': 1},
 'coef0': {'type': 'float',
          'default': 0.0,
          'transform': 'None',
          'lower': 0.0,
          'upper': 0.0},

```

```

'shrinking': {'levels': [0, 1],
'type': 'factor',
'default': 0,
'transform': 'None',
'core_model_parameter_type': 'bool',
'lower': 0,
'upper': 1},
'probability': {'levels': [0, 1],
'type': 'factor',
'default': 0,
'transform': 'None',
'core_model_parameter_type': 'bool',
'lower': 0,
'upper': 1},
'tol': {'type': 'float',
'default': 0.001,
'transform': 'None',
'lower': 0.0001,
'upper': 0.01},
'cache_size': {'type': 'float',
'default': 200,
'transform': 'None',
'lower': 100,
'upper': 400},
'break_ties': {'levels': [0, 1],
'type': 'factor',
'default': 0,
'transform': 'None',
'core_model_parameter_type': 'bool',
'lower': 0,
'upper': 1}}

```

10.6 Step 6: Modify hyper_dict Hyperparameters for the Selected Algorithm aka core_model

spotPython provides functions for modifying the hyperparameters, their bounds and factors as well as for activating and de-activating hyperparameters without re-compilation of the Python source code. These functions were described in [Section 14.6](#).

10.6.1 Modify hyperparameter of type numeric and integer (boolean)

Numeric and boolean values can be modified using the `modify_hyper_parameter_bounds` method. For example, to change the `tol` hyperparameter of the SVC model to the interval `[1e-3, 1e-2]`, the following code can be used:

```
from spotPython.hyperparameters.values import modify_hyper_parameter_bounds
fun_control = modify_hyper_parameter_bounds(fun_control, "tol", bounds=[1e-3, 1e-2])
# fun_control = modify_hyper_parameter_bounds(fun_control, "min_samples_split", bounds=[3,
#fun_control = modify_hyper_parameter_bounds(fun_control, "merit_preprune", bounds=[0, 0])
fun_control["core_model_hyper_dict"]["tol"]
```

```
{'type': 'float',
 'default': 0.001,
 'transform': 'None',
 'lower': 0.001,
 'upper': 0.01}
```

10.6.2 Modify hyperparameter of type factor

Factors can be modified with the `modify_hyper_parameter_levels` function. For example, to exclude the `sigmoid` kernel from the tuning, the `kernel` hyperparameter of the SVC model can be modified as follows:

```
from spotPython.hyperparameters.values import modify_hyper_parameter_levels
fun_control = modify_hyper_parameter_levels(fun_control, "kernel", ["linear", "poly", "rbf"])
fun_control["core_model_hyper_dict"]["kernel"]
```

```
{'levels': ['linear', 'poly', 'rbf'],
 'type': 'factor',
 'default': 'rbf',
 'transform': 'None',
 'core_model_parameter_type': 'str',
 'lower': 0,
 'upper': 2}
```

10.6.3 Optimizers

Optimizers are described in [Section 14.6.1](#).

10.7 Step 7: Selection of the Objective (Loss) Function

There are two metrics:

1. `metric_river` is used for the river based evaluation via `eval_oml_iter_progressive`.
2. `metric_sklearn` is used for the sklearn based evaluation.

```
from sklearn.metrics import mean_absolute_error, accuracy_score, roc_curve, roc_auc_score,
fun_control.update({
    "metric_sklearn": log_loss,
})
```

10.7.1 Predict Classes or Class Probabilities

If the key `"predict_proba"` is set to `True`, the class probabilities are predicted. `False` is the default, i.e., the classes are predicted.

```
fun_control.update({
    "predict_proba": False,
})
```

10.8 Step 8: Calling the SPOT Function

10.8.1 Preparing the SPOT Call

The following code passes the information about the parameter ranges and bounds to `spot`.

```
# extract the variable types, names, and bounds
from spotPython.hyperparameters.values import (get_bound_values,
    get_var_name,
    get_var_type,)
var_type = get_var_type(fun_control)
var_name = get_var_name(fun_control)
fun_control.update({"var_type": var_type,
    "var_name": var_name})
lower = get_bound_values(fun_control, "lower")
upper = get_bound_values(fun_control, "upper")
```

```
from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control))
```

name	type	default	lower	upper	transform
C	float	1.0	0.1	10	None
kernel	factor	rbf	0	2	None
degree	int	3	3	3	None
gamma	factor	scale	0	1	None
coef0	float	0.0	0	0	None
shrinking	factor	0	0	1	None
probability	factor	0	0	1	None
tol	float	0.001	0.001	0.01	None
cache_size	float	200.0	100	400	None
break_ties	factor	0	0	1	None

10.8.2 The Objective Function

The objective function is selected next. It implements an interface from `sklearn`'s training, validation, and testing methods to `spotPython`.

```
from spotPython.fun.hyper sklearn import HyperSklearn
fun = HyperSklearn().fun_sklearn
```

10.8.3 Run the Spot Optimizer

- Run SPOT for approx. x mins (`max_time`).
- Note: the run takes longer, because the evaluation time of initial design (here: `init_size`, 20 points) is not considered.

```
from spotPython.hyperparameters.values import get_default_hyperparameters_as_array
hyper_dict=SklearnHyperDict().load()
X_start = get_default_hyperparameters_as_array(fun_control, hyper_dict)
X_start
```

```
array([[1.e+00, 2.e+00, 3.e+00, 0.e+00, 0.e+00, 0.e+00, 0.e+00, 1.e-03,
        2.e+02, 0.e+00]])
```

10.8.4 Starting the Hyperparameter Tuning

```
import numpy as np
from spotPython.spot import spot
from math import inf
spot_tuner = spot.Spot(fun=fun,
                       lower = lower,
                       upper = upper,
                       fun_evals = inf,
                       fun_repeats = 1,
                       max_time = MAX_TIME,
                       noise = False,
                       tolerance_x = np.sqrt(np.spacing(1)),
                       var_type = var_type,
                       var_name = var_name,
                       infill_criterion = "y",
                       n_points = 1,
                       seed=123,
                       log_level = 50,
                       show_models= False,
                       show_progress= True,
                       fun_control = fun_control,
                       design_control={"init_size": INIT_SIZE,
                                      "repeats": 1},
                       surrogate_control={"noise": True,
                                         "cod_type": "norm",
                                         "min_theta": -4,
                                         "max_theta": 3,
                                         "n_theta": len(var_name),
                                         "model_fun_evals": 10_000,
                                         "log_level": 50
                                         })

spot_tuner.run(X_start=X_start)
```

spotPython tuning: 5.691103166702708 [#-----] 7.45%

spotPython tuning: 4.7425859722522565 [#-----] 11.16%

spotPython tuning: 4.7425859722522565 [#-----] 14.75%

spotPython tuning: 4.7425859722522565 [##-----] 18.13%

```

spotPython tuning: 4.7425859722522565 [##-----] 21.22%

spotPython tuning: 4.7425859722522565 [##-----] 23.87%

spotPython tuning: 4.7425859722522565 [###-----] 34.31%

spotPython tuning: 4.7425859722522565 [####-----] 38.08%

spotPython tuning: 4.7425859722522565 [#####-----] 52.97%

spotPython tuning: 4.7425859722522565 [#####----] 57.00%

spotPython tuning: 4.7425859722522565 [#####---] 77.81%

spotPython tuning: 4.7425859722522565 [#####--] 81.07%

spotPython tuning: 4.7425859722522565 [#####] 100.00% Done...

<spotPython.spot.spot.Spot at 0x180fed5d0>

```

10.9 Step 9: Results

```

SAVE = False
LOAD = False

if SAVE:
    result_file_name = "res_" + experiment_name + ".pkl"
    with open(result_file_name, 'wb') as f:
        pickle.dump(spot_tuner, f)

if LOAD:
    result_file_name = "res_ch10-friedman-hpt-0_maans03_60min_20init_1K_2023-04-14_10-11-1"
    with open(result_file_name, 'rb') as f:
        spot_tuner = pickle.load(f)

```

After the hyperparameter tuning run is finished, the progress of the hyperparameter tuning can be visualized. The following code generates the progress plot from `?@fig-progress`.

```
spot_tuner.plot_progress(log_y=False,
                        filename="./figures/" + experiment_name+"_progress.png")
```

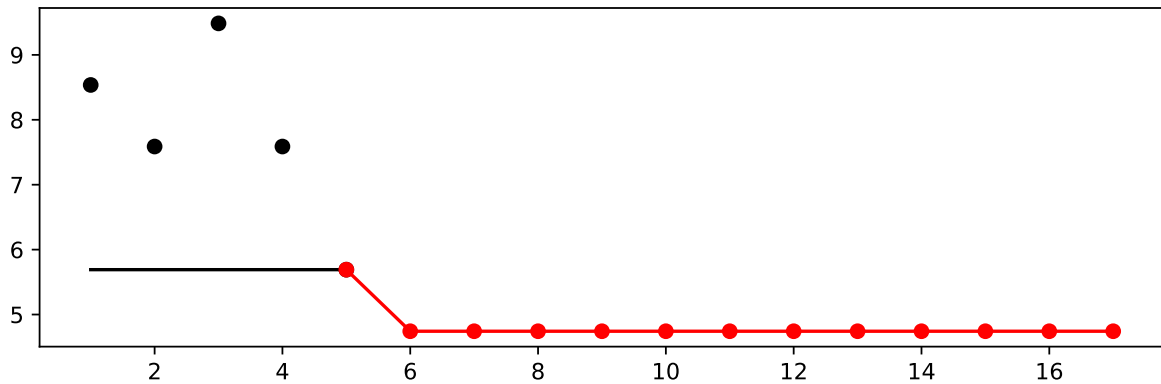


Figure 10.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- Print the results

```
print(gen_design_table(fun_control=fun_control,
                      spot=spot_tuner))
```

name	type	default	lower	upper	tuned	transform
C	float	1.0	0.1	10.0	0.23258412447782734	None
kernel	factor	rbf	0.0	2.0	1.0	None
degree	int	3	3.0	3.0	3.0	None
gamma	factor	scale	0.0	1.0	0.0	None
coef0	float	0.0	0.0	0.0	0.0	None
shrinking	factor	0	0.0	1.0	1.0	None
probability	factor	0	0.0	1.0	1.0	None
tol	float	0.001	0.001	0.01	0.003757085413122674	None
cache_size	float	200.0	100.0	400.0	214.29269330654913	None
break_ties	factor	0	0.0	1.0	1.0	None

10.9.1 Show variable importance

```
spot_tuner.plot_importance(threshold=0.025, filename="./figures/" + experiment_name+"_impo
```

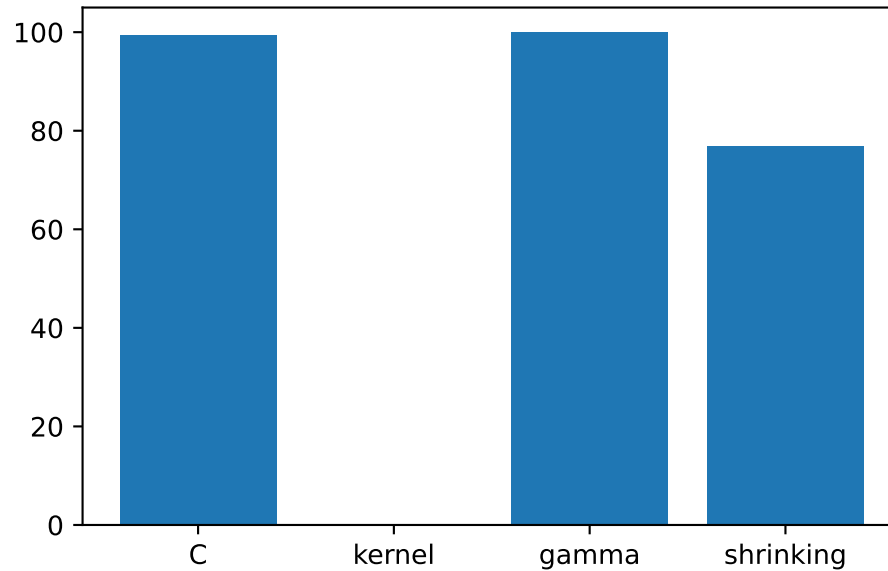


Figure 10.2: Variable importance plot, threshold 0.025.

10.9.2 Get Default Hyperparameters

```
from spotPython.hyperparameters.values import get_default_values, transform_hyper_parameter_values_default = get_default_values(fun_control) values_default = transform_hyper_parameter_values(fun_control=fun_control, hyper_parameter values_default
```

```
{'C': 1.0,
 'kernel': 'rbf',
 'degree': 3,
 'gamma': 'scale',
 'coef0': 0.0,
 'shrinking': 0,
 'probability': 0,
 'tol': 0.001,
 'cache_size': 200.0,
```

```
'break_ties': 0}
```

```
from sklearn.pipeline import make_pipeline
model_default = make_pipeline(fun_control["prep_model"], fun_control["core_model"](**value
model_default
```

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                 ('svc',
                  SVC(break_ties=0, cache_size=200.0, probability=0,
                      shrinking=0))])
```

10.9.3 Get SPOT Results

```
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
print(X)
```

```
[[2.32584124e-01 1.00000000e+00 3.00000000e+00 0.00000000e+00
 0.00000000e+00 1.00000000e+00 1.00000000e+00 3.75708541e-03
 2.14292693e+02 1.00000000e+00]]
```

```
from spotPython.hyperparameters.values import assign_values, return_conf_list_from_var_dict
v_dict = assign_values(X, fun_control["var_name"])
return_conf_list_from_var_dict(var_dict=v_dict, fun_control=fun_control)
```

```
[{'C': 0.23258412447782734,
 'kernel': 'poly',
 'degree': 3,
 'gamma': 'scale',
 'coef0': 0.0,
 'shrinking': 1,
 'probability': 1,
 'tol': 0.003757085413122674,
 'cache_size': 214.29269330654913,
 'break_ties': 1}]
```

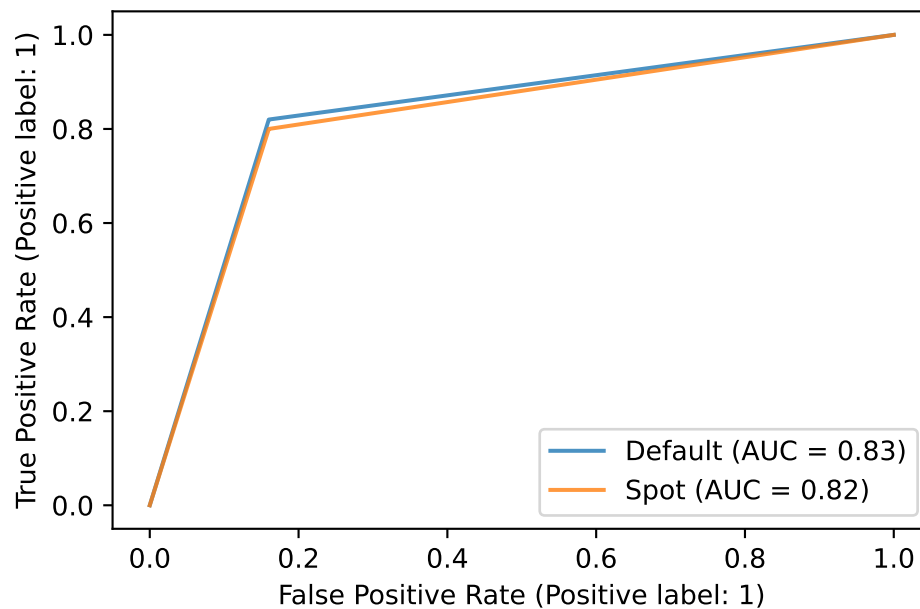
```
from spotPython.hyperparameters.values import get_one_sklearn_model_from_X
model_spot = get_one_sklearn_model_from_X(X, fun_control)
```

```
model_spot
```

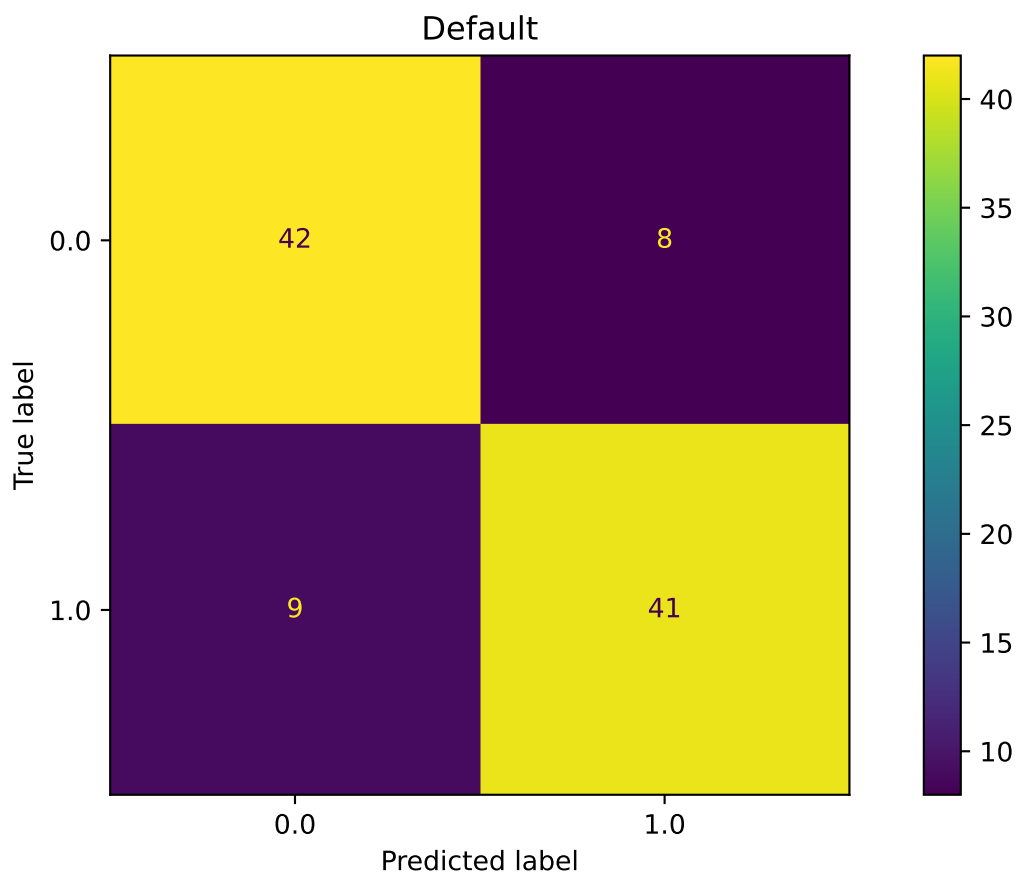
```
Pipeline(steps=[('standardscaler', StandardScaler()),  
                ('svc',  
                 SVC(C=0.23258412447782734, break_ties=1,  
                    cache_size=214.29269330654913, kernel='poly',  
                    probability=1, shrinking=1, tol=0.003757085413122674))])
```

10.9.4 Plot: Compare Predictions

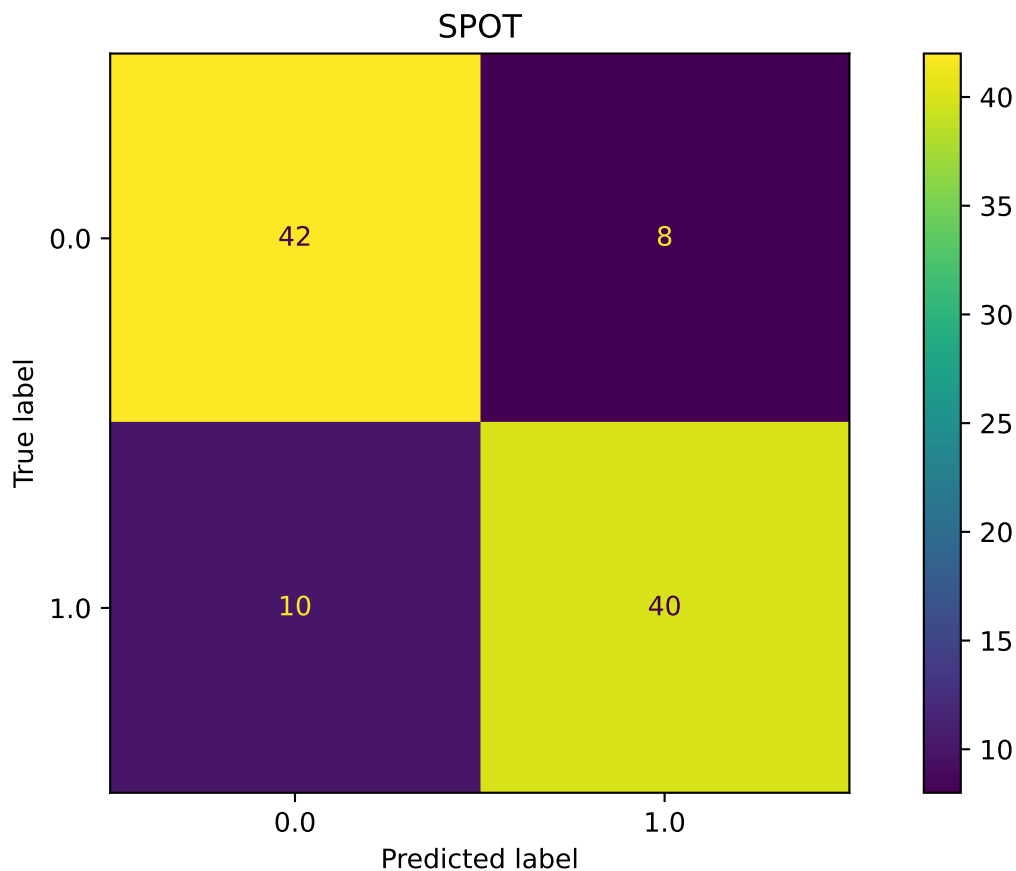
```
from spotPython.plot.validation import plot_roc  
plot_roc([model_default, model_spot], fun_control, model_names=["Default", "Spot"])
```



```
from spotPython.plot.validation import plot_confusion_matrix  
plot_confusion_matrix(model_default, fun_control, title = "Default")
```

```
plot_confusion_matrix(model_spot, fun_control, title="SPOT")
```



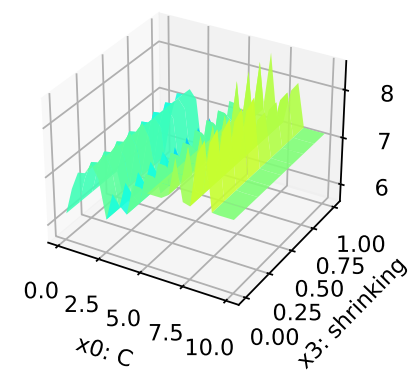
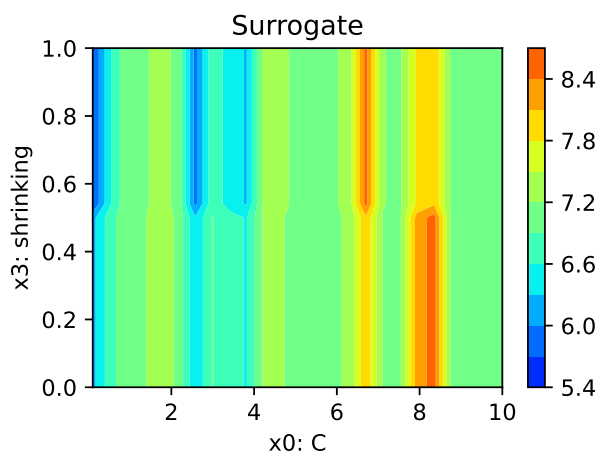
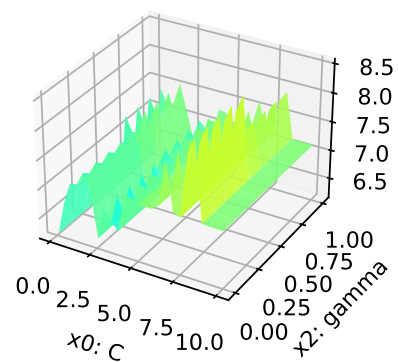
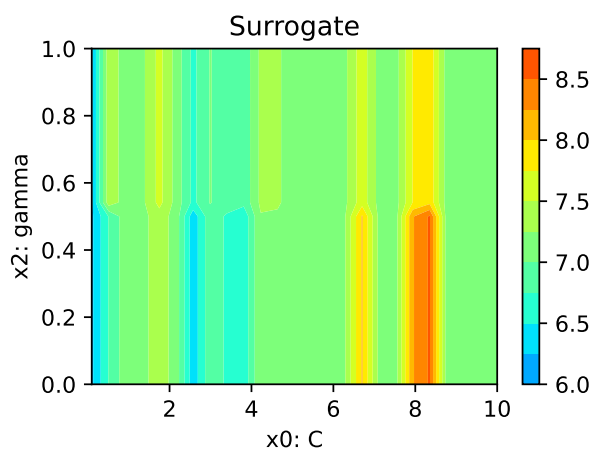
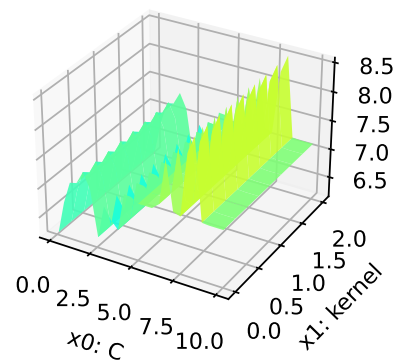
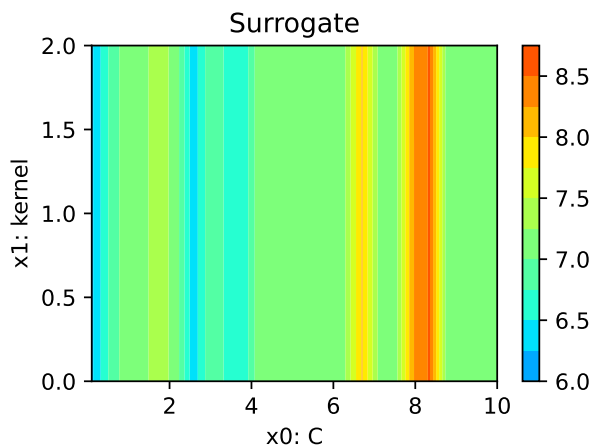
```
min(spot_tuner.y), max(spot_tuner.y)
```

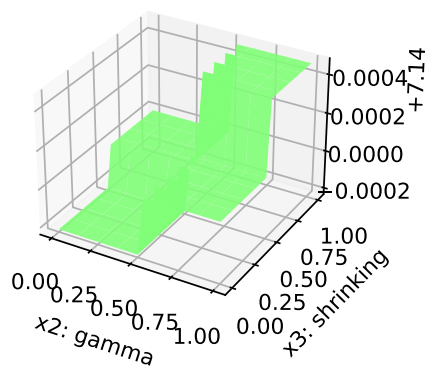
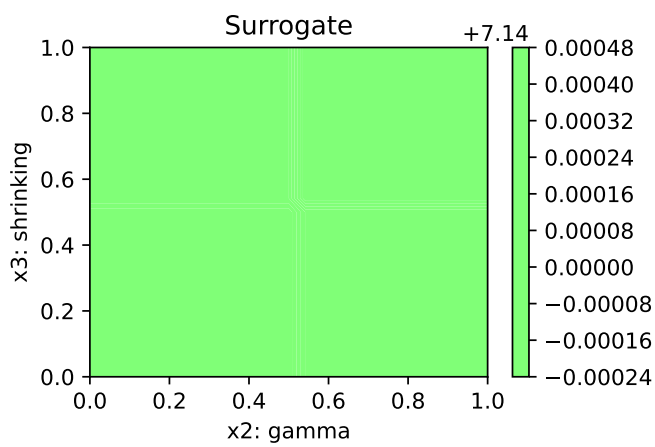
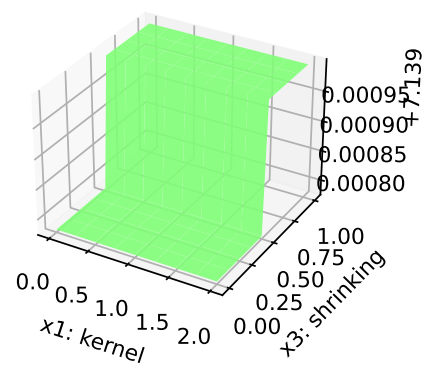
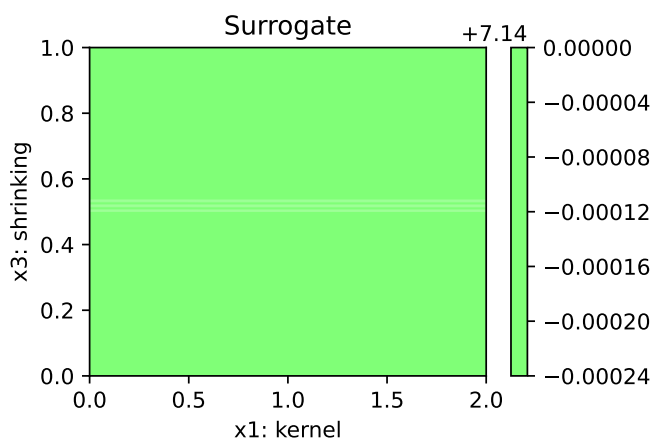
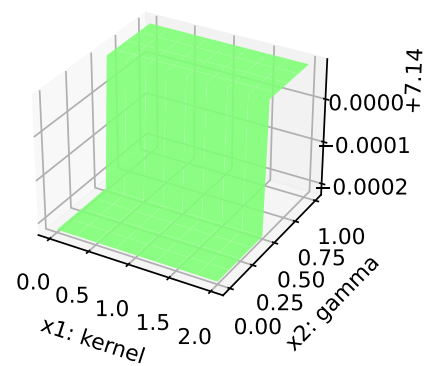
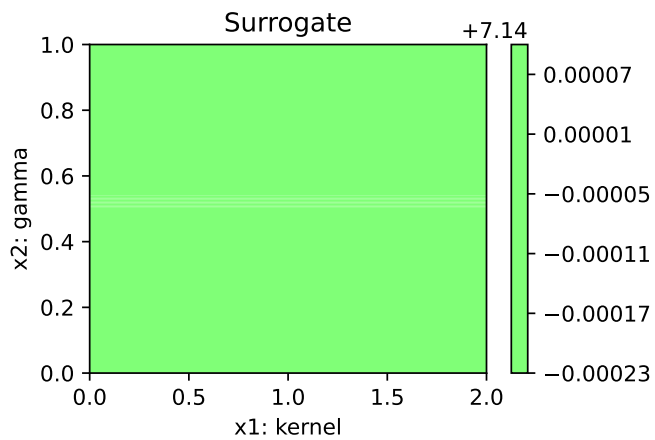
```
(4.7425859722522565, 9.485171944504513)
```

10.9.5 Detailed Hyperparameter Plots

```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

```
C: 99.24590429537938
kernel: 0.09661162966353774
gamma: 100.0
shrinking: 76.804672945321
```





10.9.6 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): text/html

10.9.7 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

```
PLOT_ALL = False
if PLOT_ALL:
    n = spot_tuner.k
    for i in range(n-1):
        for j in range(i+1, n):
            spot_tuner.plot_contour(i=i, j=j, min_z=min_z, max_z = max_z)
```

11 HPT: PyTorch With fashionMNIST

In this tutorial, we will show how `spotPython` can be integrated into the PyTorch training workflow.

This document refers to the following software versions:

- python: 3.10.10
- torch: 2.0.1
- torchvision: 0.15.0

```
pip list | grep "spot[RiverPython]"
```

spotPython	0.2.52
spotRiver	0.0.94

Note: you may need to restart the kernel to use updated packages.

`spotPython` can be installed via `pip`. Alternatively, the source code can be downloaded from `gitHub`: <https://github.com/sequential-parameter-optimization/spotPython>.

```
!pip install spotPython
```

- Uncomment the following lines if you want to for (re-)installation the latest version of `spotPython` from `gitHub`.

```
# import sys
# !{sys.executable} -m pip install --upgrade build
# !{sys.executable} -m pip install --upgrade --force-reinstall spotPython
```

11.1 Step 1: Setup

Before we consider the detailed experimental setup, we select the parameters that affect run time, initial design size and the device that is used.

🔥 Caution: Run time and initial design size should be increased for real experiments

- MAX_TIME is set to one minute for demonstration purposes. For real experiments, this should be increased to at least 1 hour.
- INIT_SIZE is set to 5 for demonstration purposes. For real experiments, this should be increased to at least 10.

i Note: Device selection

- The device can be selected by setting the variable DEVICE.
- Since we are using a simple neural net, the setting "cpu" is preferred (on Mac).
- If you have a GPU, you can use "cuda:0" instead.
- If DEVICE is set to None, spotPython will automatically select the device.
 - This might result in "mps" on Macs, which is not the best choice for simple neural nets.

```
MAX_TIME = 1
INIT_SIZE = 5
DEVICE = "cpu" # "cuda:0"
```

```
from spotPython.utils.device import getDevice
DEVICE = getDevice(DEVICE)
print(DEVICE)
```

cpu

```
import os
import copy
import socket
from datetime import datetime
from dateutil.tz import tzlocal
start_time = datetime.now(tzlocal())
HOSTNAME = socket.gethostname().split(".")[0]
experiment_name = '11-torch' + "_" + HOSTNAME + "_" + str(MAX_TIME) + "min_" + str(INIT_SIZE)
experiment_name = experiment_name.replace(':', '-')
print(experiment_name)
if not os.path.exists('./figures'):
    os.makedirs('./figures')
```

11-torch_maans03_1min_5init_2023-07-03_10-32-45

11.2 Step 2: Initialization of the Empty fun_control Dictionary

spotPython uses a Python dictionary for storing the information required for the hyperparameter tuning process, which was described in Section [14.2](#).

 Caution: Tensorboard does not work under Windows

- Since tensorboard does not work under Windows, we recommend setting the parameter `tensorboard_path` to `None` if you are working under Windows.

```
from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="classification",
    tensorboard_path="runs/11_spot_hpt_torch_fashion_mnist",
    device=DEVICE)
```

11.3 Step 3: PyTorch Data Loading

11.3.1 Load fashionMNIST Data

```
from torchvision import datasets, transforms
from torchvision.transforms import ToTensor
def load_data(data_dir="./data"):
    # Download training data from open datasets.
    training_data = datasets.FashionMNIST(
        root=data_dir,
        train=True,
        download=True,
        transform=ToTensor(),
    )
    # Download test data from open datasets.
    test_data = datasets.FashionMNIST(
        root=data_dir,
        train=False,
        download=True,
        transform=ToTensor(),
    )
    return training_data, test_data
```



```
train, test = load_data()
train.data.shape, test.data.shape
```

```
(torch.Size([60000, 28, 28]), torch.Size([10000, 28, 28]))
```

```
n_samples = len(train)
# add the dataset to the fun_control
fun_control.update({"data": None,
                   "train": train,
                   "test": test,
                   "n_samples": n_samples,
                   "target_column": None})
```

11.4 Step 4: Specification of the Preprocessing Model

After the training and test data are specified and added to the `fun_control` dictionary, `spotPython` allows the specification of a data preprocessing pipeline, e.g., for the scaling of the data or for the one-hot encoding of categorical variables, see Section 14.4. This feature is not used here, so we do not change the default value (which is `None`).

11.5 Step 5: Select Model (algorithm) and `core_model_hyper_dict`

`spotPython` implements a class which is similar to the class described in the PyTorch tutorial. The class is called `Net_fashionMNIST` and is implemented in the file `netfashionMNIST.py`. The class is imported here.

```
from torch import nn
import spotPython.torch.netcore as netcore

class Net_fashionMNIST(netcore.Net_Core):
    def __init__(self, l1, l2, lr_mult, batch_size, epochs, k_folds, patience, optimizer,
                 super(Net_fashionMNIST, self).__init__(
                     lr_mult=lr_mult,
                     batch_size=batch_size,
                     epochs=epochs,
```

```

        k_folds=k_folds,
        patience=patience,
        optimizer=optimizer,
        sgd_momentum=sgd_momentum,
    )
    self.flatten = nn.Flatten()
    self.linear_relu_stack = nn.Sequential(
        nn.Linear(28 * 28, 11),
        nn.ReLU(),
        nn.Linear(11, 12),
        nn.ReLU(),
        nn.Linear(12, 10)
    )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

```

This class inherits from the class `Net_Core` which is implemented in the file `netcore.py`, see Section 14.5.1.

```

from spotPython.data.torch_hyper_dict import TorchHyperDict
from spotPython.torch.netfashionMNIST import Net_fashionMNIST
from spotPython.hyperparameters.values import add_core_model_to_fun_control
fun_control = add_core_model_to_fun_control(core_model=Net_fashionMNIST,
                                           fun_control=fun_control,
                                           hyper_dict=TorchHyperDict,
                                           filename=None)

```

11.5.1 The Search Space

11.5.2 Configuring the Search Space With spotPython

11.5.2.1 The hyper_dict Hyperparameters for the Selected Algorithm

spotPython uses JSON files for the specification of the hyperparameters, which were described in Section 14.5.5.

The corresponding entries for the `core_model` class are shown below.

```
fun_control['core_model_hyper_dict']
```

```
{'l1': {'type': 'int',  
       'default': 5,  
       'transform': 'transform_power_2_int',  
       'lower': 2,  
       'upper': 9},  
'l2': {'type': 'int',  
       'default': 5,  
       'transform': 'transform_power_2_int',  
       'lower': 2,  
       'upper': 9},  
'lr_mult': {'type': 'float',  
            'default': 1.0,  
            'transform': 'None',  
            'lower': 0.1,  
            'upper': 10.0},  
'batch_size': {'type': 'int',  
               'default': 4,  
               'transform': 'transform_power_2_int',  
               'lower': 1,  
               'upper': 4},  
'epochs': {'type': 'int',  
            'default': 3,  
            'transform': 'transform_power_2_int',  
            'lower': 3,  
            'upper': 4},  
'k_folds': {'type': 'int',  
            'default': 1,  
            'transform': 'None',  
            'lower': 1,  
            'upper': 1},  
'patience': {'type': 'int',  
              'default': 5,  
              'transform': 'None',  
              'lower': 2,  
              'upper': 10},  
'optimizer': {'levels': ['Adadelata',  
                          'Adagrad',  
                          'Adam',  
                          'AdamW',  
                          'SparseAdam',
```

```

'Adamax',
'ASGD',
'NAdam',
'RAdam',
'RMSprop',
'Rprop',
'SGD'],
'type': 'factor',
'default': 'SGD',
'transform': 'None',
'core_model_parameter_type': 'str',
'lower': 0,
'upper': 12},
'sgd_momentum': {'type': 'float',
'default': 0.0,
'transform': 'None',
'lower': 0.0,
'upper': 1.0}}

```

11.6 Step 6: Modify hyper_dict Hyperparameters for the Selected Algorithm aka core_model

spotPython provides functions for modifying the hyperparameters, their bounds and factors as well as for activating and de-activating hyperparameters without re-compilation of the Python source code. These functions were described in Section [14.6](#).

11.6.1 Modify hyperparameter of type numeric and integer (boolean)

The hyperparameter `k_folds` is not used, it is de-activated here by setting the lower and upper bound to the same value.

 **Caution:** Small net size, number of epochs, and patience for demonstration purposes

- Net sizes 11 and 12 as well as `epochs` and `patience` are set to small values for demonstration purposes. These values are too small for a real application.
- More resonable values are, e.g.:
 - `fun_control = modify_hyper_parameter_bounds(fun_control, "11", bounds=[2, 7])`
 - `fun_control = modify_hyper_parameter_bounds(fun_control,`

```
"epochs", bounds=[7, 9]) and
- fun_control = modify_hyper_parameter_bounds(fun_control,
"patience", bounds=[2, 7])
```

```
from spotPython.hyperparameters.values import modify_hyper_parameter_bounds
fun_control = modify_hyper_parameter_bounds(fun_control, "k_folds", bounds=[0, 0])
fun_control = modify_hyper_parameter_bounds(fun_control, "patience", bounds=[2, 2])
fun_control = modify_hyper_parameter_bounds(fun_control, "epochs", bounds=[2, 3])
fun_control = modify_hyper_parameter_bounds(fun_control, "l1", bounds=[2, 5])
fun_control = modify_hyper_parameter_bounds(fun_control, "l2", bounds=[2, 5])
```

11.6.2 Modify hyperparameter of type factor

```
from spotPython.hyperparameters.values import modify_hyper_parameter_levels
fun_control = modify_hyper_parameter_levels(fun_control, "optimizer", ["Adam", "AdamW", "Ad
```

11.6.3 Optimizers

Optimizers are described in Section [14.6.1](#).

```
fun_control = modify_hyper_parameter_bounds(fun_control,
"lr_mult", bounds=[1e-3, 1e-3])
fun_control = modify_hyper_parameter_bounds(fun_control,
"sgd_momentum", bounds=[0.9, 0.9])
```

11.7 Step 7: Selection of the Objective (Loss) Function

11.7.1 Evaluation

The evaluation procedure requires the specification of two elements:

1. the way how the data is split into a train and a test set and
2. the loss function (and a metric).

These are described in Section [19.7.1](#).

The key "loss_function" specifies the loss function which is used during the optimization, see Section [14.7.5](#).

We will use CrossEntropy loss for the multiclass-classification task.

```
from torch.nn import CrossEntropyLoss
loss_function = CrossEntropyLoss()
fun_control.update({
    "loss_function": loss_function,
    "shuffle": True,
    "eval": "train_hold_out"
})
```

11.7.2 Metric

```
from torchmetrics import Accuracy
metric_torch = Accuracy(task="multiclass", num_classes=10).to(fun_control["device"])
fun_control.update({"metric_torch": metric_torch})
```

11.8 Step 8: Calling the SPOT Function

11.8.1 Preparing the SPOT Call

The following code passes the information about the parameter ranges and bounds to `spot`.

```
# extract the variable types, names, and bounds
from spotPython.hyperparameters.values import (get_bound_values,
    get_var_name,
    get_var_type,)
var_type = get_var_type(fun_control)
var_name = get_var_name(fun_control)
fun_control.update({"var_type": var_type,
    "var_name": var_name})
lower = get_bound_values(fun_control, "lower")
upper = get_bound_values(fun_control, "upper")

from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control))
```

name	type	default	lower	upper	transform
-----	-----	-----	-----	-----	-----

l1	int	5	2	5	transform_power_2_int	
l2	int	5	2	5	transform_power_2_int	
lr_mult	float	1.0	0.001	0.001	None	
batch_size	int	4	1	4	transform_power_2_int	
epochs	int	3	2	3	transform_power_2_int	
k_folds	int	1	0	0	None	
patience	int	5	2	2	None	
optimizer	factor	SGD	0	3	None	
sgd_momentum	float	0.0	0.9	0.9	None	

11.8.2 The Objective Function `fun_torch`

The objective function `fun_torch` is selected next. It implements an interface from PyTorch's training, validation, and testing methods to `spotPython`.

```
from spotPython.fun.hypertorch import HyperTorch
fun = HyperTorch().fun_torch
```

11.8.3 Starting the Hyperparameter Tuning

```
import numpy as np
from spotPython.spot import spot
from math import inf
spot_tuner = spot.Spot(fun=fun,
                      lower = lower,
                      upper = upper,
                      fun_evals = inf,
                      fun_repeats = 1,
                      max_time = MAX_TIME,
                      noise = False,
                      tolerance_x = np.sqrt(np.spacing(1)),
                      var_type = var_type,
                      var_name = var_name,
                      infill_criterion = "y",
                      n_points = 1,
                      seed=123,
                      log_level = 50,
                      show_models= False,
                      show_progress= True,
                      fun_control = fun_control,
```


MulticlassAccuracy: 0.1765416711568832 | Loss: 2.2061852587064106 | Acc: 0.1765416666666667.
Returned to Spot: Validation loss: 2.2061852587064106

config: {'l1': 8, 'l2': 8, 'lr_mult': 0.001, 'batch_size': 8, 'epochs': 4, 'k_folds': 0, 'pa
Epoch: 1 |

MulticlassAccuracy: 0.1274999976158142 | Loss: 2.3507174194653828 | Acc: 0.1275000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.1398749947547913 | Loss: 2.3380448350111642 | Acc: 0.1398750000000000.
Epoch: 3 |

MulticlassAccuracy: 0.1424166709184647 | Loss: 2.3204975862503052 | Acc: 0.1424166666666667.
Epoch: 4 |

MulticlassAccuracy: 0.1434166729450226 | Loss: 2.3063520757357279 | Acc: 0.1434166666666667.
Returned to Spot: Validation loss: 2.306352075735728

config: {'l1': 32, 'l2': 16, 'lr_mult': 0.001, 'batch_size': 2, 'epochs': 8, 'k_folds': 0, 'l
Epoch: 1 |

MulticlassAccuracy: 0.1800833344459534 | Loss: 2.1352689491907757 | Acc: 0.1800833333333333.
Epoch: 2 |

MulticlassAccuracy: 0.3207083344459534 | Loss: 1.9540089417397977 | Acc: 0.3207083333333333.
Epoch: 3 |

MulticlassAccuracy: 0.4493333399295807 | Loss: 1.7582998426010212 | Acc: 0.4493333333333333.
Epoch: 4 |

MulticlassAccuracy: 0.5227500200271606 | Loss: 1.5738397979810834 | Acc: 0.5227500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5304583311080933 | Loss: 1.4117284052881103 | Acc: 0.5304583333333334.
Epoch: 6 |

MulticlassAccuracy: 0.6135416626930237 | Loss: 1.2744226018749178 | Acc: 0.6135416666666667.
Epoch: 7 |

MulticlassAccuracy: 0.6387083530426025 | Loss: 1.1665975301762421 | Acc: 0.6387083333333333.
Epoch: 8 |

MulticlassAccuracy: 0.6391249895095825 | Loss: 1.0911609180184703 | Acc: 0.6391250000000001.
Returned to Spot: Validation loss: 1.0911609180184703

config: {'l1': 4, 'l2': 8, 'lr_mult': 0.001, 'batch_size': 4, 'epochs': 4, 'k_folds': 0, 'pa
Epoch: 1 |

MulticlassAccuracy: 0.1493333280086517 | Loss: 2.3130540801286696 | Acc: 0.1493333333333333.
Epoch: 2 |

MulticlassAccuracy: 0.1315416693687439 | Loss: 2.3043103766043980 | Acc: 0.1315416666666667.
Epoch: 3 |

MulticlassAccuracy: 0.1222083345055580 | Loss: 2.2871728708744050 | Acc: 0.1222083333333333.
Epoch: 4 |

MulticlassAccuracy: 0.1189583316445351 | Loss: 2.2711666022340458 | Acc: 0.1189583333333333.
Returned to Spot: Validation loss: 2.2711666022340458

config: {'l1': 16, 'l2': 32, 'lr_mult': 0.001, 'batch_size': 8, 'epochs': 8, 'k_folds': 0, 'p
Epoch: 1 |

MulticlassAccuracy: 0.1960833370685577 | Loss: 2.2842108801205954 | Acc: 0.1960833333333333.
Epoch: 2 |

MulticlassAccuracy: 0.2126249969005585 | Loss: 2.2636893765131632 | Acc: 0.2126250000000000.
Epoch: 3 |

MulticlassAccuracy: 0.2503750026226044 | Loss: 2.2404759271939594 | Acc: 0.2503750000000000.
Epoch: 4 |

MulticlassAccuracy: 0.3181666731834412 | Loss: 2.2149134116967519 | Acc: 0.3181666666666667.
Epoch: 5 |

MulticlassAccuracy: 0.3484166562557220 | Loss: 2.1846798245906829 | Acc: 0.3484166666666667.
Epoch: 6 |

MulticlassAccuracy: 0.3682083189487457 | Loss: 2.1514494522015255 | Acc: 0.3682083333333334.
Epoch: 7 |

MulticlassAccuracy: 0.3990833461284637 | Loss: 2.1156194982131322 | Acc: 0.3990833333333333.
Epoch: 8 |

MulticlassAccuracy: 0.4272083342075348 | Loss: 2.0771733982563019 | Acc: 0.4272083333333334.
Returned to Spot: Validation loss: 2.077173398256302

config: {'l1': 8, 'l2': 16, 'lr_mult': 0.001, 'batch_size': 8, 'epochs': 8, 'k_folds': 0, 'p
Epoch: 1 |

MulticlassAccuracy: 0.101999980926514 | Loss: 2.2929790640672048 | Acc: 0.1020000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.101999980926514 | Loss: 2.2677730861504872 | Acc: 0.1020000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.1020416691899300 | Loss: 2.2388499063650769 | Acc: 0.1020416666666667.
Epoch: 4 |

MulticlassAccuracy: 0.1022083312273026 | Loss: 2.2041082998514177 | Acc: 0.1022083333333333.
Epoch: 5 |

MulticlassAccuracy: 0.1037499979138374 | Loss: 2.1723692315419516 | Acc: 0.1037500000000000.
Epoch: 6 |

MulticlassAccuracy: 0.1137916669249535 | Loss: 2.1327384761174519 | Acc: 0.1137916666666667.
Epoch: 7 |

MulticlassAccuracy: 0.1447083353996277 | Loss: 2.0856389589309692 | Acc: 0.1447083333333333.
Epoch: 8 |

MulticlassAccuracy: 0.1749999970197678 | Loss: 2.0325956790447237 | Acc: 0.1750000000000000.
Returned to Spot: Validation loss: 2.0325956790447237

spotPython tuning: 1.0911609180184703 [#####] 100.00% Done...

<spotPython.spot.spot.Spot at 0x1889bfb20>

11.9 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section 14.9, see also the description in the documentation: [Tensorboard](#).

11.10 Step 10: Results

After the hyperparameter tuning run is finished, the results can be analyzed as described in Section 14.10.

```
SAVE = False
LOAD = False

if SAVE:
    result_file_name = "res_" + experiment_name + ".pkl"
    with open(result_file_name, 'wb') as f:
        pickle.dump(spot_tuner, f)

if LOAD:
    result_file_name = "ADD THE NAME here, e.g.: res_ch10-friedman-hpt-0_maans03_60min_20i"
    with open(result_file_name, 'rb') as f:
        spot_tuner = pickle.load(f)
```

After the hyperparameter tuning run is finished, the progress of the hyperparameter tuning can be visualized. The following code generates the progress plot from `?@fig-progress`.

```
spot_tuner.plot_progress(log_y=False,
    filename="./figures/" + experiment_name+"_progress.png")
```

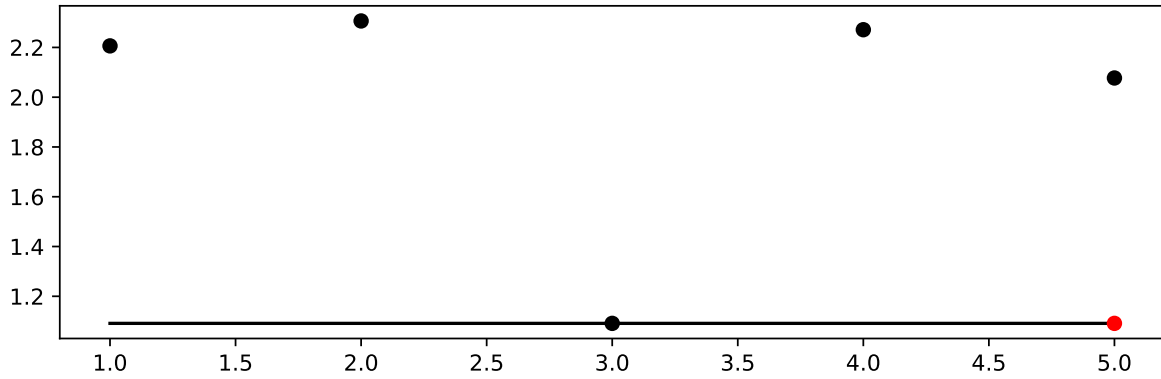


Figure 11.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- Print the results

```
print(gen_design_table(fun_control=fun_control,
                      spot=spot_tuner))
```

name	type	default	lower	upper	tuned	transform
l1	int	5	2.0	5.0	5.0	transform_power_2_int
l2	int	5	2.0	5.0	4.0	transform_power_2_int
lr_mult	float	1.0	0.001	0.001	0.001	None
batch_size	int	4	1.0	4.0	1.0	transform_power_2_int
epochs	int	3	2.0	3.0	3.0	transform_power_2_int
k_folds	int	1	0.0	0.0	0.0	None
patience	int	5	2.0	2.0	2.0	None
optimizer	factor	SGD	0.0	3.0	3.0	None
sgd_momentum	float	0.0	0.9	0.9	0.9	None

11.10.1 Show variable importance

```
spot_tuner.plot_importance(threshold=0.025, filename="./figures/" + experiment_name+"_impo
```

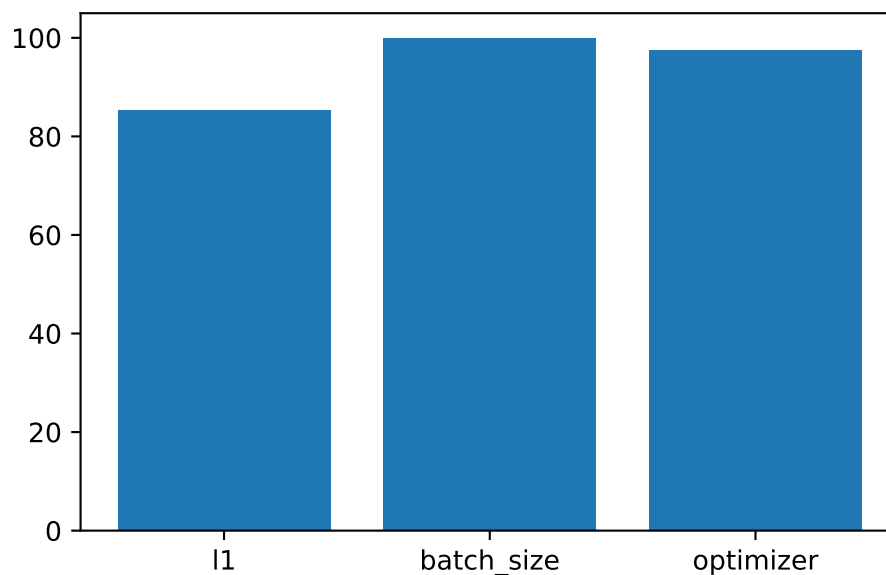


Figure 11.2: Variable importance plot, threshold 0.025.

11.10.2 Get the Tuned Architecture (SPOT Results)

The architecture of the `spotPython` model can be obtained by the following code:

```
from spotPython.hyperparameters.values import get_one_core_model_from_X
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
model_spot = get_one_core_model_from_X(X, fun_control)
model_spot
```

```
Net_fashionMNIST(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=32, bias=True)
    (1): ReLU()
    (2): Linear(in_features=32, out_features=16, bias=True)
    (3): ReLU()
    (4): Linear(in_features=16, out_features=10, bias=True)
  )
)
```

11.10.3 Get Default Hyperparameters

```
fc = fun_control
fc.update({"core_model_hyper_dict":
          hyper_dict[fun_control["core_model"].__name__]})
model_default = get_one_core_model_from_X(X_start, fun_control=fc)
model_default
```

```
Net_fashionMNIST(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=32, bias=True)
    (1): ReLU()
    (2): Linear(in_features=32, out_features=32, bias=True)
    (3): ReLU()
    (4): Linear(in_features=32, out_features=10, bias=True)
  )
)
```

11.10.4 Evaluation of the Default and the Tuned Architectures

The method `train_tuned` takes a model architecture without trained weights and trains this model with the train data. The train data is split into train and validation data. The validation data is used for early stopping. The trained model weights are saved as a dictionary.

```
from spotPython.torch.traintest import train_tuned
train_tuned(net=model_default, train_dataset=train, shuffle=True,
            loss_function=fun_control["loss_function"],
            metric=fun_control["metric_torch"],
            device = fun_control["device"],
            show_batch_interval=1_000_000,
            path=None,
            task=fun_control["task"])
```

Epoch: 1 |

MulticlassAccuracy: 0.3722916543483734 | Loss: 2.0754299880663556 | Acc: 0.3722916666666667.
Epoch: 2 |

MulticlassAccuracy: 0.5687916874885559 | Loss: 1.5182934633096059 | Acc: 0.5687916666666667.
Epoch: 3 |

MulticlassAccuracy: 0.6010416746139526 | Loss: 1.2225897694428762 | Acc: 0.6010416666666667.
Epoch: 4 |

MulticlassAccuracy: 0.6252083182334900 | Loss: 1.0736844333807627 | Acc: 0.6252083333333334.
Epoch: 5 |

MulticlassAccuracy: 0.6464166641235352 | Loss: 0.9859142975608508 | Acc: 0.6464166666666666.
Epoch: 6 |

MulticlassAccuracy: 0.6554999947547913 | Loss: 0.9291729355653127 | Acc: 0.6555000000000000.
Epoch: 7 |

MulticlassAccuracy: 0.6671666502952576 | Loss: 0.8874406706293424 | Acc: 0.6671666666666667.
Epoch: 8 |

MulticlassAccuracy: 0.6760833263397217 | Loss: 0.8565445574323336 | Acc: 0.6760833333333334.
Returned to Spot: Validation loss: 0.8565445574323336

```
from spotPython.torch.traintest import test_tuned
test_tuned(net=model_default, test_dataset=test,
            loss_function=fun_control["loss_function"],
            metric=fun_control["metric_torch"],
            shuffle=False,
            device = fun_control["device"],
            task=fun_control["task"])
```

MulticlassAccuracy: 0.6665999889373779 | Loss: 0.8699883942604065 | Acc: 0.6666000000000000.
Final evaluation: Validation loss: 0.8699883942604065
Final evaluation: Validation metric: 0.6665999889373779

(0.8699883942604065, nan, tensor(0.6666))

The following code trains the model `model_spot`. If `path` is set to a filename, e.g., `path = "model_spot_trained.pt"`, the weights of the trained model will be saved to this file.


```
train_tuned(net=model_spot, train_dataset=train,
            loss_function=fun_control["loss_function"],
            metric=fun_control["metric_torch"],
            shuffle=True,
            device = fun_control["device"],
            path=None,
            task=fun_control["task"])
```

Epoch: 1 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.257

MulticlassAccuracy: 0.2839166522026062 | Loss: 2.1027970579663910 | Acc: 0.2839166666666667.
Epoch: 2 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.030

MulticlassAccuracy: 0.4356666803359985 | Loss: 1.8297560552557310 | Acc: 0.4356666666666666.
Epoch: 3 |

Batch: 10000. Batch Size: 2. Training Loss (running): 1.748

MulticlassAccuracy: 0.5072083473205566 | Loss: 1.5589698385695616 | Acc: 0.5072083333333334.
Epoch: 4 |

Batch: 10000. Batch Size: 2. Training Loss (running): 1.491

MulticlassAccuracy: 0.5603333115577698 | Loss: 1.3440139626041054 | Acc: 0.5603333333333333.
Epoch: 5 |

Batch: 10000. Batch Size: 2. Training Loss (running): 1.299

MulticlassAccuracy: 0.5910000205039978 | Loss: 1.1893416738472880 | Acc: 0.5910000000000000.
Epoch: 6 |

Batch: 10000. Batch Size: 2. Training Loss (running): 1.156

MulticlassAccuracy: 0.6155416369438171 | Loss: 1.0787782829242447 | Acc: 0.6155416666666667.
Epoch: 7 |

Batch: 10000. Batch Size: 2. Training Loss (running): 1.055

MulticlassAccuracy: 0.6355833411216736 | Loss: 1.0008049506886552 | Acc: 0.6355833333333333.
Epoch: 8 |

Batch: 10000. Batch Size: 2. Training Loss (running): 0.985

MulticlassAccuracy: 0.6532916426658630 | Loss: 0.9433055154327303 | Acc: 0.6532916666666667.
Returned to Spot: Validation loss: 0.9433055154327303

```
test_tuned(net=model_spot, test_dataset=test,
            shuffle=False,
            loss_function=fun_control["loss_function"],
            metric=fun_control["metric_torch"],
            device = fun_control["device"],
            task=fun_control["task"])
```

MulticlassAccuracy: 0.6470000147819519 | Loss: 0.9550195944413543 | Acc: 0.6470000000000000.
Final evaluation: Validation loss: 0.9550195944413543
Final evaluation: Validation metric: 0.6470000147819519

(0.9550195944413543, nan, tensor(0.6470))

11.10.5 Detailed Hyperparameter Plots

```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

l1: 85.24670570169687
batch_size: 100.0
optimizer: 97.43970323096576

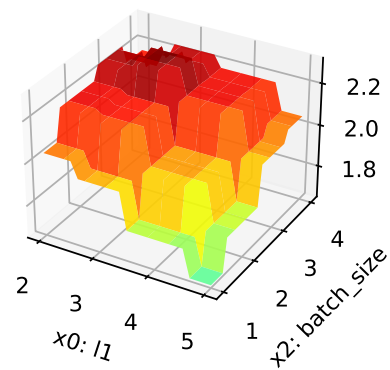
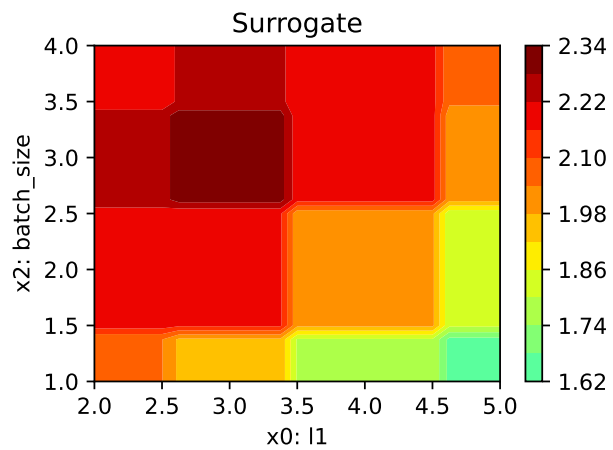
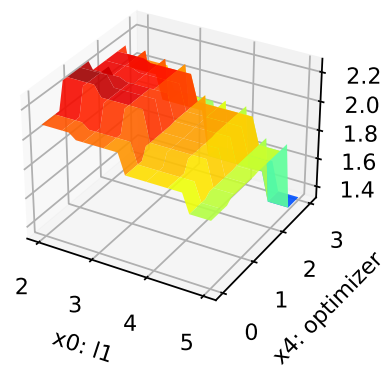
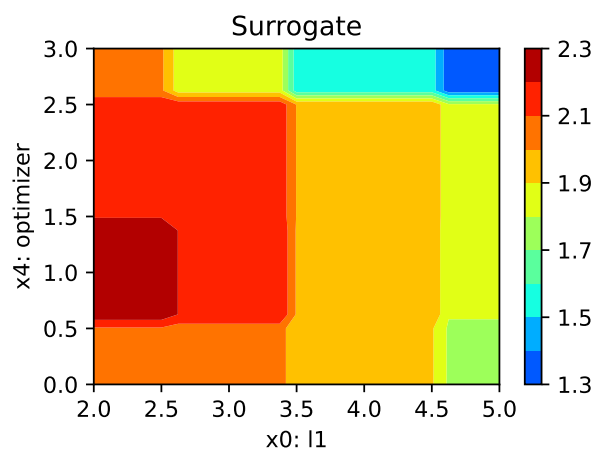
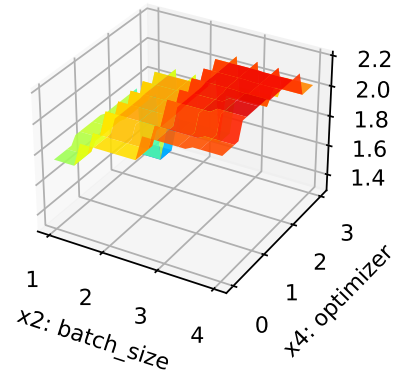
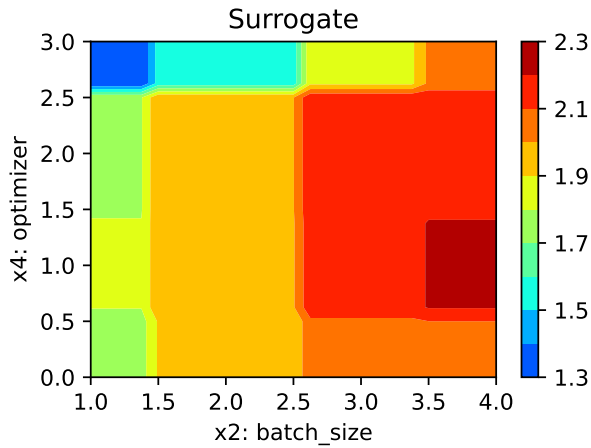


Figure 11.3: Contour plots.





11.10.6 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

Unable to display output for mime type(s): text/html

Parallel coordinates plots

Unable to display output for mime type(s): text/html

11.10.7 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

```
PLOT_ALL = False
if PLOT_ALL:
    n = spot_tuner.k
    for i in range(n-1):
        for j in range(i+1, n):
            spot_tuner.plot_contour(i=i, j=j, min_z=min_z, max_z = max_z)
```

12 HPT: PyTorch With cifar10 Data

In this tutorial, we will show how `spotPython` can be integrated into the PyTorch training workflow.

This document refers to the following software versions:

- python: 3.10.10
- torch: 2.0.1
- torchvision: 0.15.0

```
pip list | grep "spot[RiverPython]"
```

<code>spotPython</code>	0.2.52
<code>spotRiver</code>	0.0.94

Note: you may need to restart the kernel to use updated packages.

`spotPython` can be installed via `pip`. Alternatively, the source code can be downloaded from `gitHub`: <https://github.com/sequential-parameter-optimization/spotPython>.

```
!pip install spotPython
```

- Uncomment the following lines if you want to for (re-)installation the latest version of `spotPython` from `gitHub`.

```
# import sys
# !{sys.executable} -m pip install --upgrade build
# !{sys.executable} -m pip install --upgrade --force-reinstall spotPython
```

12.1 Step 1: Setup

Before we consider the detailed experimental setup, we select the parameters that affect run time, initial design size and the device that is used.

🔥 Caution: Run time and initial design size should be increased for real experiments

- MAX_TIME is set to one minute for demonstration purposes. For real experiments, this should be increased to at least 1 hour.
- INIT_SIZE is set to 5 for demonstration purposes. For real experiments, this should be increased to at least 10.

i Note: Device selection

- The device can be selected by setting the variable DEVICE.
- Since we are using a simple neural net, the setting "cpu" is preferred (on Mac).
- If you have a GPU, you can use "cuda:0" instead.
- If DEVICE is set to None, spotPython will automatically select the device.
 - This might result in "mps" on Macs, which is not the best choice for simple neural nets.

```
MAX_TIME = 1
INIT_SIZE = 5
DEVICE = "cpu" # "cuda:0" None
```

```
from spotPython.utils.device import getDevice
DEVICE = getDevice(DEVICE)
print(DEVICE)
```

cpu

```
import os
import copy
import socket
from datetime import datetime
from dateutil.tz import tzlocal
start_time = datetime.now(tzlocal())
HOSTNAME = socket.gethostname().split(".")[0]
experiment_name = '12-torch' + "_" + HOSTNAME + "_" + str(MAX_TIME) + "min_" + str(INIT_SIZE)
experiment_name = experiment_name.replace(':', '-')
print(experiment_name)
if not os.path.exists('./figures'):
    os.makedirs('./figures')
```

12-torch_maans03_1min_5init_2023-07-03_11-12-07

12.2 Step 2: Initialization of the Empty fun_control Dictionary

spotPython uses a Python dictionary for storing the information required for the hyperparameter tuning process, which was described in [Section 14.2](#).

 Caution: Tensorboard does not work under Windows

- Since tensorboard does not work under Windows, we recommend setting the parameter `tensorboard_path` to `None` if you are working under Windows.

```
from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="classification",
    tensorboard_path="runs/12_spot_hpt_torch_cifar10",
    device=DEVICE)
```

12.3 Step 3: PyTorch Data Loading

12.3.1 Load Data Cifar10 Data

```
from torchvision import datasets, transforms
import torchvision
def load_data(data_dir="./data"):
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

    trainset = torchvision.datasets.CIFAR10(
        root=data_dir, train=True, download=True, transform=transform)

    testset = torchvision.datasets.CIFAR10(
        root=data_dir, train=False, download=True, transform=transform)

    return trainset, testset
train, test = load_data()
```

Files already downloaded and verified

Files already downloaded and verified

- Since this works fine, we can add the data loading to the `fun_control` dictionary:

```
n_samples = len(train)
# add the dataset to the fun_control
fun_control.update({"data": None, # dataset,
                  "train": train,
                  "test": test,
                  "n_samples": n_samples,
                  "target_column": None})
```

12.4 Step 4: Specification of the Preprocessing Model

After the training and test data are specified and added to the `fun_control` dictionary, `spotPython` allows the specification of a data preprocessing pipeline, e.g., for the scaling of the data or for the one-hot encoding of categorical variables, see Section 14.4. This feature is not used here, so we do not change the default value (which is `None`).

12.5 Step 5: Select Model (algorithm) and `core_model_hyper_dict`

12.5.1 Implementing a Configurable Neural Network With `spotPython`

`spotPython` includes the `Net_CIFAR10` class which is implemented in the file `netcifar10.py`. The class is imported here.

This class inherits from the class `Net_Core` which is implemented in the file `netcore.py`, see Section 14.5.1.

```
from spotPython.torch.netcifar10 import Net_CIFAR10
from spotPython.data.torch_hyper_dict import TorchHyperDict
from spotPython.hyperparameters.values import add_core_model_to_fun_control
fun_control = add_core_model_to_fun_control(core_model=Net_CIFAR10,
                                          fun_control=fun_control,
                                          hyper_dict=TorchHyperDict,
                                          filename=None)
```


12.5.2 The Search Space

12.5.3 Configuring the Search Space With spotPython

12.5.3.1 The `hyper_dict` Hyperparameters for the Selected Algorithm

spotPython uses JSON files for the specification of the hyperparameters, which were described in Section 14.5.5.

The corresponding entries for the `core_model` class are shown below.

```
fun_control['core_model_hyper_dict']

{'l1': {'type': 'int',
       'default': 5,
       'transform': 'transform_power_2_int',
       'lower': 2,
       'upper': 9},
 'l2': {'type': 'int',
       'default': 5,
       'transform': 'transform_power_2_int',
       'lower': 2,
       'upper': 9},
 'lr_mult': {'type': 'float',
            'default': 1.0,
            'transform': 'None',
            'lower': 0.1,
            'upper': 10.0},
 'batch_size': {'type': 'int',
               'default': 4,
               'transform': 'transform_power_2_int',
               'lower': 1,
               'upper': 4},
 'epochs': {'type': 'int',
            'default': 3,
            'transform': 'transform_power_2_int',
            'lower': 3,
            'upper': 4},
 'k_folds': {'type': 'int',
            'default': 1,
            'transform': 'None',
            'lower': 1,
```

```

    'upper': 1},
'patience': {'type': 'int',
'default': 5,
'transform': 'None',
'lower': 2,
'upper': 10},
'optimizer': {'levels': ['Adadelata',
    'Adagrad',
    'Adam',
    'AdamW',
    'SparseAdam',
    'Adamax',
    'ASGD',
    'NAdam',
    'RAdam',
    'RMSprop',
    'Rprop',
    'SGD'],
'type': 'factor',
'default': 'SGD',
'transform': 'None',
'class_name': 'torch.optim',
'core_model_parameter_type': 'str',
'lower': 0,
'upper': 12},
'sgd_momentum': {'type': 'float',
'default': 0.0,
'transform': 'None',
'lower': 0.0,
'upper': 1.0}}

```

12.6 Step 6: Modify hyper_dict Hyperparameters for the Selected Algorithm aka core_model

spotPython provides functions for modifying the hyperparameters, their bounds and factors as well as for activating and de-activating hyperparameters without re-compilation of the Python source code. These functions were described in [Section 14.6](#).

12.6.1 Step 5: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model`

12.6.1.1 Modify Hyperparameters of Type numeric and integer (boolean)

The hyperparameter `k_folds` is not used, it is de-activated here by setting the lower and upper bound to the same value.

 Caution: Small net size, number of epochs, and patience for demonstration purposes

- Net sizes 11 and 12 as well as `epochs` and `patience` are set to small values for demonstration purposes. These values are too small for a real application.
- More resonable values are, e.g.:

```
– fun_control = modify_hyper_parameter_bounds(fun_control, "l1",
    bounds=[2, 7])
– fun_control = modify_hyper_parameter_bounds(fun_control,
    "epochs", bounds=[7, 9]) and
– fun_control = modify_hyper_parameter_bounds(fun_control,
    "patience", bounds=[2, 7])
```

```
from spotPython.hyperparameters.values import modify_hyper_parameter_bounds
fun_control = modify_hyper_parameter_bounds(fun_control, "k_folds", bounds=[0, 0])
fun_control = modify_hyper_parameter_bounds(fun_control, "patience", bounds=[2, 2])
fun_control = modify_hyper_parameter_bounds(fun_control, "epochs", bounds=[2, 3])
fun_control = modify_hyper_parameter_bounds(fun_control, "l1", bounds=[2, 5])
fun_control = modify_hyper_parameter_bounds(fun_control, "l2", bounds=[2, 5])
```

12.6.2 Modify hyperparameter of type factor

```
from spotPython.hyperparameters.values import modify_hyper_parameter_levels
fun_control = modify_hyper_parameter_levels(fun_control, "optimizer", ["Adam", "AdamW", "Ad
```

12.6.3 Optimizers

Optimizers can be selected as described in Section [19.6.2](#).

Optimizers are described in Section [14.6.1](#).

```

fun_control = modify_hyper_parameter_bounds(fun_control,
    "lr_mult", bounds=[1e-3, 1e-3])
fun_control = modify_hyper_parameter_bounds(fun_control,
    "sgd_momentum", bounds=[0.9, 0.9])

```

12.7 Step 7: Selection of the Objective (Loss) Function

12.7.1 Evaluation

The evaluation procedure requires the specification of two elements:

1. the way how the data is split into a train and a test set and
2. the loss function (and a metric).

These are described in Section [19.7.1](#).

The key "loss_function" specifies the loss function which is used during the optimization, see Section [14.7.5](#).

We will use CrossEntropy loss for the multiclass-classification task.

```

from torch.nn import CrossEntropyLoss
loss_function = CrossEntropyLoss()
fun_control.update({
    "loss_function": loss_function,
    "shuffle": True,
    "eval": "train_hold_out"
})

```

12.7.2 Metric

```

import torchmetrics
metric_torch = torchmetrics.Accuracy(task="multiclass",
    num_classes=10).to(fun_control["device"])
fun_control.update({"metric_torch": metric_torch})

```

12.8 Step 8: Calling the SPOT Function

12.8.1 Preparing the SPOT Call

The following code passes the information about the parameter ranges and bounds to `spot`.

```
# extract the variable types, names, and bounds
from spotPython.hyperparameters.values import (get_bound_values,
        get_var_name,
        get_var_type,)
var_type = get_var_type(fun_control)
var_name = get_var_name(fun_control)
fun_control.update({"var_type": var_type,
                   "var_name": var_name})
lower = get_bound_values(fun_control, "lower")
upper = get_bound_values(fun_control, "upper")

from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control))
```

name	type	default	lower	upper	transform
l1	int	5	2	5	transform_power_2_int
l2	int	5	2	5	transform_power_2_int
lr_mult	float	1.0	0.001	0.001	None
batch_size	int	4	1	4	transform_power_2_int
epochs	int	3	2	3	transform_power_2_int
k_folds	int	1	0	0	None
patience	int	5	2	2	None
optimizer	factor	SGD	0	3	None
sgd_momentum	float	0.0	0.9	0.9	None

12.8.2 The Objective Function `fun_torch`

The objective function `fun_torch` is selected next. It implements an interface from PyTorch's training, validation, and testing methods to `spotPython`.

```
from spotPython.fun.hypertorch import HyperTorch
fun = HyperTorch().fun_torch
```


MulticlassAccuracy: 0.1004000008106232 | Loss: 2.3213387151718141 | Acc: 0.1004000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.1004000008106232 | Loss: 2.3194895351409914 | Acc: 0.1004000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.1004000008106232 | Loss: 2.3174200672149659 | Acc: 0.1004000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.1004000008106232 | Loss: 2.3150155567169191 | Acc: 0.1004000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.1004000008106232 | Loss: 2.3125426223754881 | Acc: 0.1004000000000000.
Epoch: 7 |

MulticlassAccuracy: 0.1004000008106232 | Loss: 2.3103920515060423 | Acc: 0.1004000000000000.
Epoch: 8 |

MulticlassAccuracy: 0.1004500016570091 | Loss: 2.3083329513549806 | Acc: 0.1004500000000000.
Returned to Spot: Validation loss: 2.3083329513549806

config: {'l1': 8, 'l2': 8, 'lr_mult': 0.001, 'batch_size': 8, 'epochs': 4, 'k_folds': 0, 'pa
Epoch: 1 |

MulticlassAccuracy: 0.0987500026822090 | Loss: 2.3179213603019715 | Acc: 0.0987500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.1002999991178513 | Loss: 2.3168445756912233 | Acc: 0.1003000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.1064499989151955 | Loss: 2.3159697061538695 | Acc: 0.1064500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.1145000010728836 | Loss: 2.3151374659538271 | Acc: 0.1145000000000000.
Returned to Spot: Validation loss: 2.315137465953827

config: {'l1': 32, 'l2': 16, 'lr_mult': 0.001, 'batch_size': 2, 'epochs': 8, 'k_folds': 0, 'p
Epoch: 1 |

MulticlassAccuracy: 0.1028499975800514 | Loss: 2.3065456705808640 | Acc: 0.1028500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.1173999980092049 | Loss: 2.2966628431320188 | Acc: 0.1174000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.1528999954462051 | Loss: 2.2584952401041987 | Acc: 0.1529000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.1580500006675720 | Loss: 2.2108294310688974 | Acc: 0.1580500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.1589999943971634 | Loss: 2.1711846857666970 | Acc: 0.1590000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.1640499979257584 | Loss: 2.1371860899329187 | Acc: 0.1640500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.1751500070095062 | Loss: 2.1060472553730012 | Acc: 0.1751500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.1858499944210052 | Loss: 2.0774075149416924 | Acc: 0.1858500000000000.
Returned to Spot: Validation loss: 2.0774075149416924

config: {'l1': 4, 'l2': 8, 'lr_mult': 0.001, 'batch_size': 4, 'epochs': 4, 'k_folds': 0, 'pa
Epoch: 1 |

MulticlassAccuracy: 0.1001499965786934 | Loss: 2.3315476094722749 | Acc: 0.1001500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.1001499965786934 | Loss: 2.3303059405803679 | Acc: 0.1001500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.1001499965786934 | Loss: 2.3289187388181687 | Acc: 0.1001500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.1001000031828880 | Loss: 2.3274010416507722 | Acc: 0.1001000000000000.
Returned to Spot: Validation loss: 2.327401041650772

config: {'l1': 16, 'l2': 32, 'lr_mult': 0.001, 'batch_size': 8, 'epochs': 8, 'k_folds': 0, 'p
Epoch: 1 |

MulticlassAccuracy: 0.0927999988198280 | Loss: 2.3063064372062683 | Acc: 0.0928000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.0933500006794930 | Loss: 2.3060600970268248 | Acc: 0.0933500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.0934500023722649 | Loss: 2.3057814753532409 | Acc: 0.0934500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.0935499966144562 | Loss: 2.3054392121315002 | Acc: 0.0935500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.0942000001668930 | Loss: 2.3050192249298096 | Acc: 0.0942000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.0950499996542931 | Loss: 2.3045036982536318 | Acc: 0.0950500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.0962499976158142 | Loss: 2.3038450118064882 | Acc: 0.0962500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.1012500002980232 | Loss: 2.3029967868804930 | Acc: 0.1012500000000000.
Returned to Spot: Validation loss: 2.302996786880493

config: {'l1': 8, 'l2': 16, 'lr_mult': 0.001, 'batch_size': 8, 'epochs': 8, 'k_folds': 0, 'p
Epoch: 1 |

MulticlassAccuracy: 0.1009000018239021 | Loss: 2.3312099172592164 | Acc: 0.1009000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.1009000018239021 | Loss: 2.3279800788879395 | Acc: 0.1009000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.1109500005841255 | Loss: 2.3245738817214967 | Acc: 0.1109500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.1382499933242798 | Loss: 2.3212142275810241 | Acc: 0.1382500000000000.
Epoch: 5 |

```
MulticlassAccuracy: 0.1424999982118607 | Loss: 2.3181941164970397 | Acc: 0.1425000000000000.  
Epoch: 6 |
```

```
MulticlassAccuracy: 0.1428000032901764 | Loss: 2.3150831417083739 | Acc: 0.1428000000000000.  
Epoch: 7 |
```

```
MulticlassAccuracy: 0.1416500061750412 | Loss: 2.3113945014953612 | Acc: 0.1416500000000000.  
Epoch: 8 |
```

```
MulticlassAccuracy: 0.1405500024557114 | Loss: 2.3069930368423464 | Acc: 0.1405500000000000.  
Returned to Spot: Validation loss: 2.3069930368423464  
spotPython tuning: 2.0774075149416924 [#####] 100.00% Done...
```

```
<spotPython.spot.spot.Spot at 0x18c12fb20>
```

12.9 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section 14.9, see also the description in the documentation: [Tensorboard](#).

12.10 Step 10: Results

After the hyperparameter tuning run is finished, the results can be analyzed as described in Section 14.10.

```
SAVE = False  
LOAD = False  
  
if SAVE:  
    result_file_name = "res_" + experiment_name + ".pkl"  
    with open(result_file_name, 'wb') as f:  
        pickle.dump(spot_tuner, f)  
  
if LOAD:  
    result_file_name = "ADD THE NAME here, e.g.: res_ch10-friedman-hpt-0_maans03_60min_20i  
    with open(result_file_name, 'rb') as f:  
        spot_tuner = pickle.load(f)
```

After the hyperparameter tuning run is finished, the progress of the hyperparameter tuning can be visualized. The following code generates the progress plot from `?@fig-progress`.

```
spot_tuner.plot_progress(log_y=False,
    filename="./figures/" + experiment_name+"_progress.png")
```

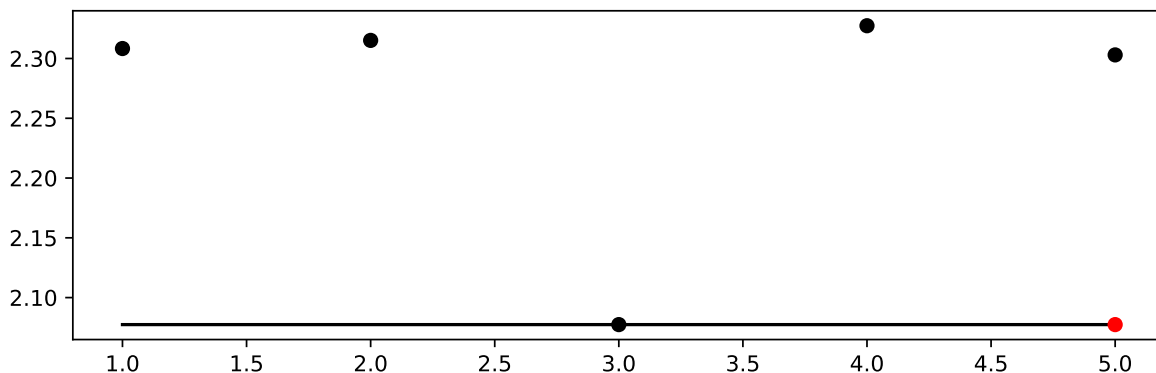


Figure 12.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- Print the results

```
print(gen_design_table(fun_control=fun_control,
    spot=spot_tuner))
```

name	type	default	lower	upper	tuned	transform
l1	int	5	2.0	5.0	5.0	transform_power_2_int
l2	int	5	2.0	5.0	4.0	transform_power_2_int
lr_mult	float	1.0	0.001	0.001	0.001	None
batch_size	int	4	1.0	4.0	1.0	transform_power_2_int
epochs	int	3	2.0	3.0	3.0	transform_power_2_int
k_folds	int	1	0.0	0.0	0.0	None
patience	int	5	2.0	2.0	2.0	None
optimizer	factor	SGD	0.0	3.0	3.0	None
sgd_momentum	float	0.0	0.9	0.9	0.9	None

12.10.1 Show variable importance

```
spot_tuner.plot_importance(threshold=0.025, filename="./figures/" + experiment_name+"_impo
```



Figure 12.2: Variable importance plot, threshold 0.025.

12.10.2 Get the Tuned Architecture (SPOT Results)

The architecture of the `spotPython` model can be obtained by the following code:

```
from spotPython.hyperparameters.values import get_one_core_model_from_X
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
model_spot = get_one_core_model_from_X(X, fun_control)
model_spot
```

```
Net_CIFAR10(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=32, bias=True)
  (fc2): Linear(in_features=32, out_features=16, bias=True)
  (fc3): Linear(in_features=16, out_features=10, bias=True)
)
```

12.10.3 Evaluation of the Tuned Architecture

```
from spotPython.torch.traintest import (
    train_tuned,
    test_tuned,
)

train_tuned(net=model_spot, train_dataset=train,
            loss_function=fun_control["loss_function"],
            metric=fun_control["metric_torch"],
            shuffle=True,
            device = fun_control["device"],
            path=None,
            task=fun_control["task"],)
```

Epoch: 1 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.310

MulticlassAccuracy: 0.1036000028252602 | Loss: 2.3006046615839004 | Acc: 0.1036000000000000.
Epoch: 2 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.292

MulticlassAccuracy: 0.1433500051498413 | Loss: 2.2680888182759285 | Acc: 0.1433500000000000.
Epoch: 3 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.257

MulticlassAccuracy: 0.1770499944686890 | Loss: 2.2302701745629312 | Acc: 0.1770500000000000.
Epoch: 4 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.224

MulticlassAccuracy: 0.1736499965190887 | Loss: 2.1976760300278664 | Acc: 0.1736500000000000.
Epoch: 5 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.195

MulticlassAccuracy: 0.1816000044345856 | Loss: 2.1689404532074930 | Acc: 0.1816000000000000.
Epoch: 6 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.164

MulticlassAccuracy: 0.1942500025033951 | Loss: 2.1427407929718494 | Acc: 0.1942500000000000.
Epoch: 7 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.145

MulticlassAccuracy: 0.2108999937772751 | Loss: 2.1194734202206136 | Acc: 0.2109000000000000.
Epoch: 8 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.117

MulticlassAccuracy: 0.222699998092651 | Loss: 2.0977164102852344 | Acc: 0.2227000000000000.
Returned to Spot: Validation loss: 2.0977164102852344

If `path` is set to a filename, e.g., `path = "model_spot_trained.pt"`, the weights of the trained model will be loaded from this file.

```
test_tuned(net=model_spot, test_dataset=test,
            shuffle=False,
            loss_function=fun_control["loss_function"],
            metric=fun_control["metric_torch"],
            device = fun_control["device"],
            task=fun_control["task"],)
```

MulticlassAccuracy: 0.2190999984741211 | Loss: 2.0944503646016122 | Acc: 0.2191000000000000.
Final evaluation: Validation loss: 2.094450364601612
Final evaluation: Validation metric: 0.2190999984741211

(2.094450364601612, nan, tensor(0.2191))

12.10.4 Cross-validated Evaluations

Caution: Cross-validated Evaluations

- The number of folds is set to 1 by default.
- Here it was changed to 3 for demonstration purposes.
- Set the number of folds to a reasonable value, e.g., 10.
- This can be done by setting the `k_folds` attribute of the model as follows:
- `setattr(model_spot, "k_folds", 10)`

```
from spotPython.torch.traintest import evaluate_cv
# modify k-folds:
setattr(model_spot, "k_folds", 3)
df_eval, df_preds, df_metrics = evaluate_cv(net=model_spot,
                                             dataset=fun_control["data"],
                                             loss_function=fun_control["loss_function"],
                                             metric=fun_control["metric_torch"],
                                             task=fun_control["task"],
                                             writer=fun_control["writer"],
                                             writerId="model_spot_cv",
                                             device = fun_control["device"])
```

Error in Net_Core. Call to evaluate_cv() failed. err=TypeError("Expected sequence or array-like")

```
metric_name = type(fun_control["metric_torch"]).__name__
print(f"loss: {df_eval}, Cross-validated {metric_name}: {df_metrics}")
```

loss: nan, Cross-validated MulticlassAccuracy: nan

12.10.5 Detailed Hyperparameter Plots

```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

batch_size: 99.99999999999999

12.10.6 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

Unable to display output for mime type(s): text/html

Parallel coordinates plots

Unable to display output for mime type(s): text/html

12.10.7 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

```
PLOT_ALL = False
if PLOT_ALL:
    n = spot_tuner.k
    for i in range(n-1):
        for j in range(i+1, n):
            spot_tuner.plot_contour(i=i, j=j, min_z=min_z, max_z = max_z)
```


13 HPT: River

River is a Python library for online machine learning (Montiel et al. 2021). It aims to be the most user-friendly library for doing machine learning on streaming data. River is the result of a merger between creme and scikit-multiflow.

13.1 Step 1: Setup

Before we consider the detailed experimental setup, we select the parameters that affect run time, initial design size and the device that is used.

 **Caution:** Run time and initial design size should be increased for real experiments

- MAX_TIME is set to one minute for demonstration purposes. For real experiments, this should be increased to at least 1 hour.
- INIT_SIZE is set to 5 for demonstration purposes. For real experiments, this should be increased to at least 10.
- K is set to 0.1 for demonstration purposes. For real experiments, this should be increased to at least 1.

```
MAX_TIME = 1
INIT_SIZE = 5
K = .1
```

10-river_maans03_1min_5init_2023-07-03_11-40-01

13.1.1 river Hyperparameter Tuning: HATR with Friedman Drift Data

- This notebook exemplifies hyperparameter tuning with SPOT (spotPython and spotRiver).
- The hyperparameter software SPOT was developed in R (statistical programming language), see Open Access book “Hyperparameter Tuning for Machine and Deep Learning with R - A Practical Guide”, available here: <https://link.springer.com/book/10.1007/978-981-19-5170-1>.

- This notebook demonstrates hyperparameter tuning for `river`. It is based on the notebook “Incremental decision trees in river: the Hoeffding Tree case”, see: <https://riverml.xyz/0.15.0/recipes/on-hoeffding-trees/#42-regression-tree-splitters>.
- Here we will use the river HTR and HATR functions as in “Incremental decision trees in river: the Hoeffding Tree case”, see: <https://riverml.xyz/0.15.0/recipes/on-hoeffding-trees/#42-regression-tree-splitters>.

```
pip list | grep "spot[RiverPython]"
```

```
spotPython          0.2.52
spotRiver           0.0.94
```

Note: you may need to restart the kernel to use updated packages.

```
# import sys
# !{sys.executable} -m pip install --upgrade build
# !{sys.executable} -m pip install --upgrade --force-reinstall spotPython
```

13.2 Step 2: Initialization of the `fun_control` Dictionary

```
from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="regression",
    tensorboard_path=None)
```

13.3 Step 3: Load the Friedman Drift Data

```
horizon = 7*24
k = K
n_total = int(k*100_000)
n_samples = n_total
p_1 = int(k*25_000)
p_2 = int(k*50_000)
position=(p_1, p_2)
n_train = 1_000
a = n_train + p_1 - 12
b = a + 12
```

- Since we also need a `river` version of the data below for plotting the model, the corresponding data set is generated here. Note: `spotRiver` uses the `train` and `test` data sets, while `river` uses the `X` and `y` data sets

```
from river.datasets import synth
import pandas as pd
dataset = synth.FriedmanDrift(
    drift_type='gra',
    position=position,
    seed=123
)
data_dict = {key: [] for key in list(dataset.take(1))[0][0].keys()}
data_dict["y"] = []
for x, y in dataset.take(n_total):
    for key, value in x.items():
        data_dict[key].append(value)
    data_dict["y"].append(y)
df = pd.DataFrame(data_dict)
# Add column names x1 until x10 to the first 10 columns of the dataframe and the column name y
df.columns = [f"x{i}" for i in range(1, 11)] + ["y"]

train = df[:n_train]
test = df[n_train:]
target_column = "y"
#
fun_control.update({"data": None, # dataset,
                  "train": train,
                  "test": test,
                  "n_samples": n_samples,
                  "target_column": target_column})
```

13.4 Step 4: Specification of the Preprocessing Model

```
from river import preprocessing
prep_model = preprocessing.StandardScaler()
fun_control.update({"prep_model": prep_model})
```

13.5 Step 5: Select algorithm and core_model_hyper_dict

- The `river` model (HATR) is selected.
- Furthermore, the corresponding hyperparameters, see: <https://riverml.xyz/0.15.0/api/tree/HoeffdingTreeRegressor/> are selected (incl. type information, names, and bounds).
- The corresponding hyperparameter dictionary is added to the `fun_control` dictionary.
- Alternatively, you can load a local `hyper_dict`. Simply set `river_hyper_dict.json` as the filename. If `filename` is set to `None`, the `hyper_dict` is loaded from the `spotRiver` package.

```
from river.tree import HoeffdingAdaptiveTreeRegressor
from spotRiver.data.river_hyper_dict import RiverHyperDict
from spotPython.hyperparameters.values import add_core_model_to_fun_control
core_model = HoeffdingAdaptiveTreeRegressor
fun_control = add_core_model_to_fun_control(core_model=core_model,
                                           fun_control=fun_control,
                                           hyper_dict=RiverHyperDict,
                                           filename=None)
```

The corresponding entries for the `core_model` class are shown below.

```
fun_control['core_model_hyper_dict']
```

```
{'grace_period': {'type': 'int',
                  'default': 200,
                  'transform': 'None',
                  'lower': 10,
                  'upper': 1000},
 'max_depth': {'type': 'int',
                'default': 20,
                'transform': 'transform_power_2_int',
                'lower': 2,
                'upper': 20},
 'delta': {'type': 'float',
            'default': 1e-07,
            'transform': 'None',
            'lower': 1e-08,
            'upper': 1e-06},
 'tau': {'type': 'float',
          'default': 0.05,
          'transform': 'None',
          'lower': 0.01,
```

```

    'upper': 0.1},
'leaf_prediction': {'levels': ['mean', 'model', 'adaptive'],
    'type': 'factor',
    'default': 'mean',
    'transform': 'None',
    'core_model_parameter_type': 'str',
    'lower': 0,
    'upper': 2},
'leaf_model': {'levels': ['LinearRegression', 'PAREgressor', 'Perceptron'],
    'type': 'factor',
    'default': 'LinearRegression',
    'transform': 'None',
    'class_name': 'river.linear_model',
    'core_model_parameter_type': 'instance()',
    'lower': 0,
    'upper': 2},
'model_selector_decay': {'type': 'float',
    'default': 0.95,
    'transform': 'None',
    'lower': 0.9,
    'upper': 0.99},
'splitter': {'levels': ['EBSTSplitter', 'TEBSTSplitter', 'QOSplitter'],
    'type': 'factor',
    'default': 'EBSTSplitter',
    'transform': 'None',
    'class_name': 'river.tree.splitter',
    'core_model_parameter_type': 'instance()',
    'lower': 0,
    'upper': 2},
'min_samples_split': {'type': 'int',
    'default': 5,
    'transform': 'None',
    'lower': 2,
    'upper': 10},
'bootstrap_sampling': {'levels': [0, 1],
    'type': 'factor',
    'default': 0,
    'transform': 'None',
    'core_model_parameter_type': 'bool',
    'lower': 0,
    'upper': 1},
'drift_window_threshold': {'type': 'int',
    'default': 300,

```

```

'transform': 'None',
'lower': 100,
'upper': 500},
'switch_significance': {'type': 'float',
'default': 0.05,
'transform': 'None',
'lower': 0.01,
'upper': 0.1},
'binary_split': {'levels': [0, 1],
'type': 'factor',
'default': 0,
'transform': 'None',
'core_model_parameter_type': 'bool',
'lower': 0,
'upper': 1},
'max_size': {'type': 'float',
'default': 500.0,
'transform': 'None',
'lower': 100.0,
'upper': 1000.0},
'memory_estimate_period': {'type': 'int',
'default': 1000000,
'transform': 'None',
'lower': 100000,
'upper': 1000000},
'stop_mem_management': {'levels': [0, 1],
'type': 'factor',
'default': 0,
'transform': 'None',
'core_model_parameter_type': 'bool',
'lower': 0,
'upper': 1},
'remove_poor_attrs': {'levels': [0, 1],
'type': 'factor',
'default': 0,
'transform': 'None',
'core_model_parameter_type': 'bool',
'lower': 0,
'upper': 1},
'merit_preprune': {'levels': [0, 1],
'type': 'factor',
'default': 0,
'transform': 'None',

```

```
'core_model_parameter_type': 'bool',
'lower': 0,
'upper': 1}}
```

13.6 Step 6: Modify hyper_dict Hyperparameters for the Selected Algorithm aka core_model

13.6.1 Modify hyperparameter of type factor

```
# fun_control = modify_hyper_parameter_levels(fun_control, "leaf_model", ["LinearRegression", "LogisticRegression"])
# fun_control["core_model_hyper_dict"]
```

13.6.2 Modify hyperparameter of type numeric and integer (boolean)

```
from spotPython.hyperparameters.values import modify_hyper_parameter_bounds
fun_control = modify_hyper_parameter_bounds(fun_control, "delta", bounds=[1e-10, 1e-6])
# fun_control = modify_hyper_parameter_bounds(fun_control, "min_samples_split", bounds=[3, 10])
fun_control = modify_hyper_parameter_bounds(fun_control, "merit_preprune", [0, 0])
```

13.7 Step 7: Selection of the Objective (Loss) Function

There are three metrics:

1. ``metric_river`` is used for the river based evaluation via ``eval_oml_iter_progressive``.
2. ``metric_sklearn`` is used for the sklearn based evaluation via ``eval_oml_horizon``.
3. ``metric_torch`` is used for the pytorch based evaluation.

```
import numpy as np
from river import metrics
from sklearn.metrics import mean_absolute_error

from spotRiver.fun.hyperriver import HyperRiver
fun = HyperRiver(seed=123, log_level=50).fun_oml_horizon
weights = np.array([1, 1/1000, 1/1000])*10_000.0
horizon = 7*24
```

```

oml_grace_period = 2
step = 100
weight_coeff = 1.0

fun_control.update({
    "horizon": horizon,
    "oml_grace_period": oml_grace_period,
    "weights": weights,
    "step": step,
    "log_level": 50,
    "weight_coeff": weight_coeff,
    "metric_river": metrics.MAE(),
    "metric_sklearn": mean_absolute_error
})

```

13.8 Step 8: Calling the SPOT Function

13.8.1 Prepare the SPOT Parameters

- Get types and variable names as well as lower and upper bounds for the hyperparameters.

```

from spotPython.hyperparameters.values import (
    get_var_type,
    get_var_name,
    get_bound_values
)

var_type = get_var_type(fun_control)
var_name = get_var_name(fun_control)
fun_control.update({"var_type": var_type,
                    "var_name": var_name})

lower = get_bound_values(fun_control, "lower")
upper = get_bound_values(fun_control, "upper")

from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control))

```

name	type	default	lower	upper	transform
-----	-----	-----	-----	-----	-----

grace_period	int	200		10	1000	None
max_depth	int	20		2	20	transform_pow
delta	float	1e-07		1e-10	1e-06	None
tau	float	0.05		0.01	0.1	None
leaf_prediction	factor	mean		0	2	None
leaf_model	factor	LinearRegression		0	2	None
model_selector_decay	float	0.95		0.9	0.99	None
splitter	factor	EBSTSplitter		0	2	None
min_samples_split	int	5		2	10	None
bootstrap_sampling	factor	0		0	1	None
drift_window_threshold	int	300		100	500	None
switch_significance	float	0.05		0.01	0.1	None
binary_split	factor	0		0	1	None
max_size	float	500.0		100	1000	None
memory_estimate_period	int	1000000		100000	1e+06	None
stop_mem_management	factor	0		0	1	None
remove_poor_attrs	factor	0		0	1	None
merit_preprune	factor	0		0	0	None

13.8.2 Run the Spot Optimizer

- Run SPOT for approx. x mins (max_time).
- Note: the run takes longer, because the evaluation time of initial design (here: initi_size, 20 points) is not considered.

```
from spotPython.hyperparameters.values import get_default_hyperparameters_as_array
hyper_dict=RiverHyperDict().load()
X_start = get_default_hyperparameters_as_array(fun_control, hyper_dict)
```

```
from spotPython.spot import spot
from math import inf
import numpy as np
spot_tuner = spot.Spot(fun=fun,
                      lower = lower,
                      upper = upper,
                      fun_evals = inf,
                      fun_repeats = 1,
                      max_time = MAX_TIME,
                      noise = False,
                      tolerance_x = np.sqrt(np.spacing(1)),
                      var_type = var_type,
```

```

        var_name = var_name,
        infill_criterion = "y",
        n_points = 1,
        seed=123,
        log_level = 50,
        show_models= False,
        show_progress= True,
        fun_control = fun_control,
        design_control={"init_size": INIT_SIZE,
                        "repeats": 1},
        surrogate_control={"noise": True,
                           "cod_type": "norm",
                           "min_theta": -4,
                           "max_theta": 3,
                           "n_theta": len(var_name),
                           "model_fun_evals": 10_000,
                           "log_level": 50
                          })

spot_tuner.run(X_start=X_start)

```

spotPython tuning: 2.2211924973008017 [#####----] 56.07%

spotPython tuning: 2.2211924973008017 [#####] 100.00% Done...

<spotPython.spot.spot.Spot at 0x189c3b010>

13.9 Step 9: Results

```

import pickle
SAVE = False
LOAD = False

if SAVE:
    result_file_name = "res_" + experiment_name + ".pkl"
    with open(result_file_name, 'wb') as f:
        pickle.dump(spot_tuner, f)

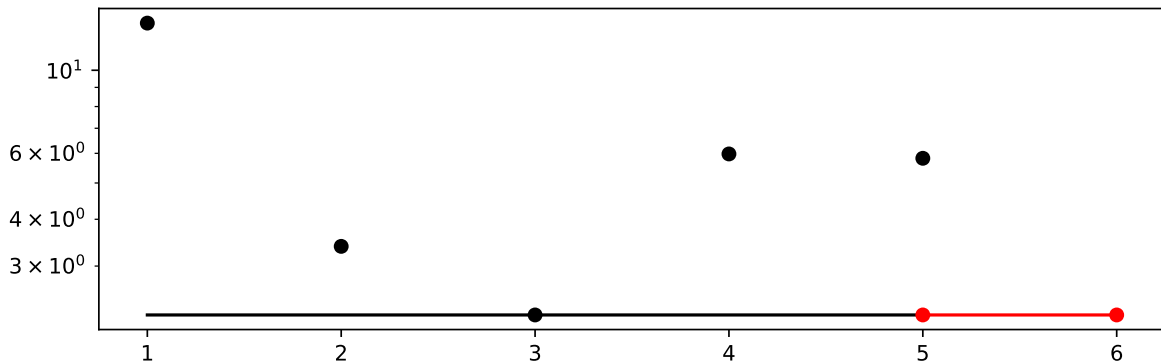
if LOAD:
    result_file_name = "res_ch10-friedman-hpt-0_maans03_60min_20init_1K_2023-04-14_10-11-1

```

```
with open(result_file_name, 'rb') as f:
    spot_tuner = pickle.load(f)
```

- Show the Progress of the hyperparameter tuning:

```
spot_tuner.plot_progress(log_y=True, filename="./figures/" + experiment_name+"_progress.pdf")
```



- Print the Results

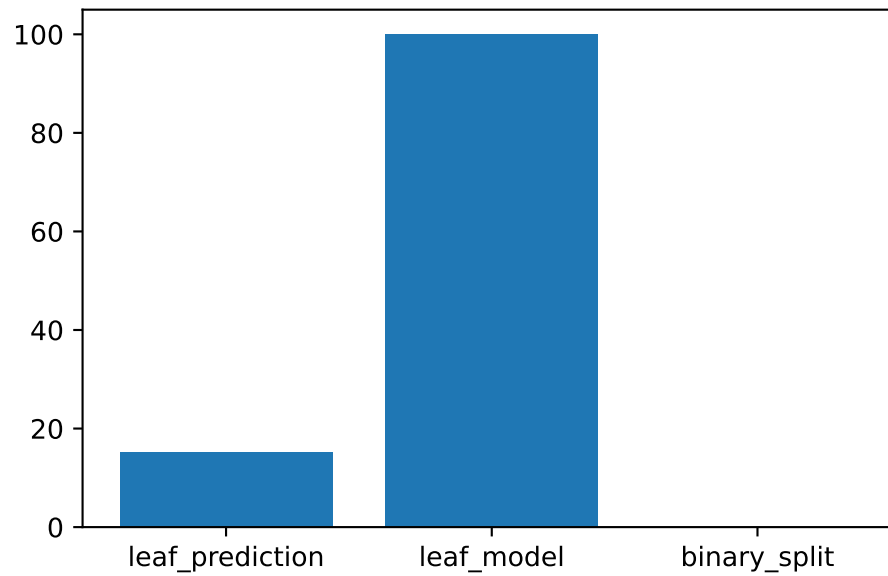
```
print(gen_design_table(fun_control=fun_control, spot=spot_tuner))
```

name	type	default	lower	upper	
grace_period	int	200	10.0	1000.0	
max_depth	int	20	2.0	20.0	
delta	float	1e-07	1e-10	1e-06	4.068723023437
tau	float	0.05	0.01	0.1	0.0484260091
leaf_prediction	factor	mean	0.0	2.0	
leaf_model	factor	LinearRegression	0.0	2.0	
model_selector_decay	float	0.95	0.9	0.99	0.970713237
splitter	factor	EBSTSplitter	0.0	2.0	
min_samples_split	int	5	2.0	10.0	
bootstrap_sampling	factor	0	0.0	1.0	
drift_window_threshold	int	300	100.0	500.0	
switch_significance	float	0.05	0.01	0.1	0.040370639
binary_split	factor	0	0.0	1.0	
max_size	float	500.0	100.0	1000.0	454.140654
memory_estimate_period	int	1000000	100000.0	1000000.0	9
stop_mem_management	factor	0	0.0	1.0	

remove_poor_attrs	factor 0	0.0	1.0
merit_preprune	factor 0	0.0	0.0

13.9.1 Show variable importance

```
spot_tuner.plot_importance(threshold=0.0025, filename="./figures/" + experiment_name+"_imp
```



13.9.2 Build and Evaluate HTR Model with Tuned Hyperparameters

```
m = test.shape[0]
a = int(m/2)-50
b = int(m/2)
```

13.9.3 The Large Data Set (k=0.2)

Caution: Increased Friedman-Drift Data Set

- The Friedman-Drift Data Set is increased by a factor of two to show the transferability of the hyperparameter tuning results.
- Larger values of k lead to a longer run time.

```
horizon = 7*24
k = .2
n_total = int(k*100_000)
n_samples = n_total
p_1 = int(k*25_000)
p_2 = int(k*50_000)
position=(p_1, p_2)
n_train = 1_000
a = n_train + p_1 - 12
b = a + 12
dataset = synth.FriedmanDrift(
    drift_type='gra',
    position=position,
    seed=123
)
data_dict = {key: [] for key in list(dataset.take(1))[0][0].keys()}
data_dict["y"] = []
for x, y in dataset.take(n_total):
    for key, value in x.items():
        data_dict[key].append(value)
    data_dict["y"].append(y)
df = pd.DataFrame(data_dict)
# Add column names x1 until x10 to the first 10 columns of the dataframe and the column name y
df.columns = [f"x{i}" for i in range(1, 11)] + ["y"]

train = df[:n_train]
test = df[n_train:]
target_column = "y"
#
fun_control.update({"data": None, # dataset,
                   "train": train,
                   "test": test,
                   "n_samples": n_samples,
```

```
"target_column": target_column})
```

13.9.4 Get Default Hyperparameters

```
# fun_control was modified, we generate a new one with the original
# default hyperparameters
from spotPython.hyperparameters.values import get_one_core_model_from_X
fc = fun_control
fc.update({"core_model_hyper_dict":
    hyper_dict[fun_control["core_model"].__name__]})
model_default = get_one_core_model_from_X(X_start, fun_control=fc)
model_default
```

```
HoeffdingAdaptiveTreeRegressor (
  grace_period=200
  max_depth=1048576
  delta=1e-07
  tau=0.05
  leaf_prediction="mean"
  leaf_model=LinearRegression (
    optimizer=SGD (
      lr=Constant (
        learning_rate=0.01
      )
    )
    loss=Squared ()
    l2=0.
    l1=0.
    intercept_init=0.
    intercept_lr=Constant (
      learning_rate=0.01
    )
    clip_gradient=1e+12
    initializer=Zeros ()
  )
  model_selector_decay=0.95
  nominal_attributes=None
  splitter=EBSTSplitter ()
  min_samples_split=5
  bootstrap_sampling=0
```

```

drift_window_threshold=300
drift_detector=ADWIN (
    delta=0.002
    clock=32
    max_buckets=5
    min_window_length=5
    grace_period=10
)
switch_significance=0.05
binary_split=0
max_size=500.
memory_estimate_period=1000000
stop_mem_management=0
remove_poor_attrs=0
merit_preprune=0
seed=None
)

```

```

from spotRiver.evaluation.eval_bml import eval_oml_horizon

```

```

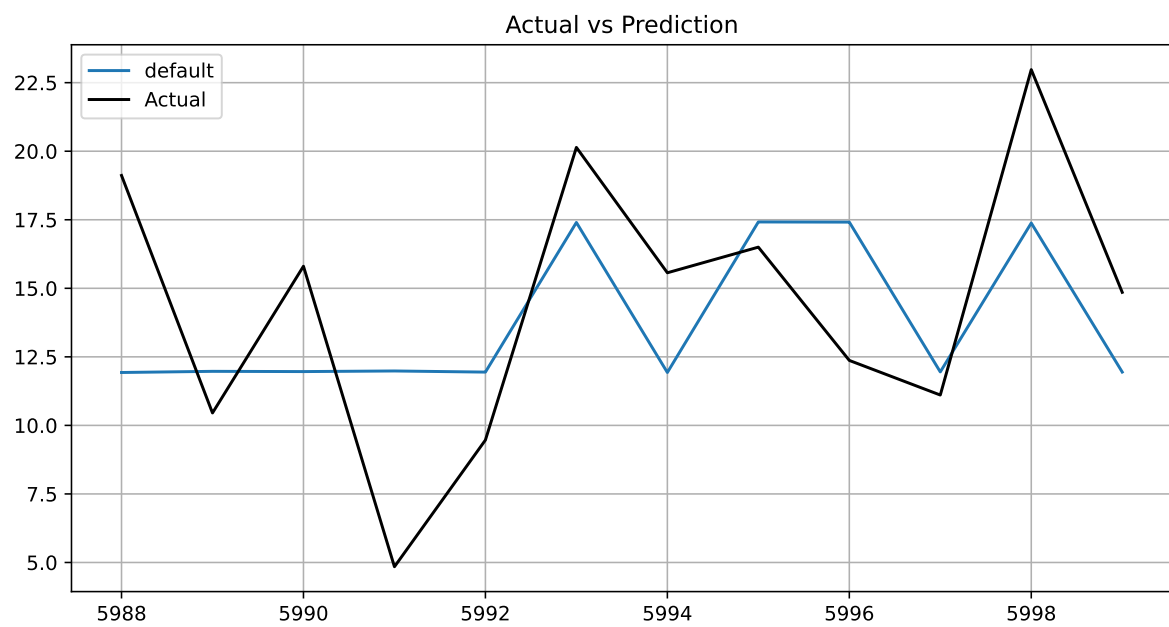
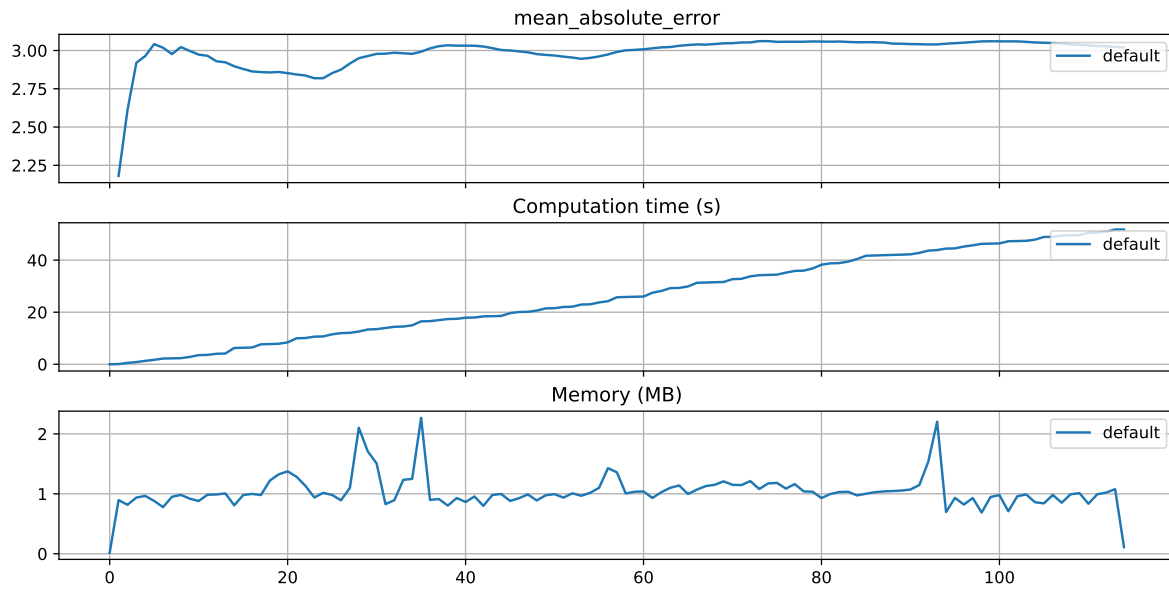
df_eval_default, df_true_default = eval_oml_horizon(
    model=model_default,
    train=fun_control["train"],
    test=fun_control["test"],
    target_column=fun_control["target_column"],
    horizon=fun_control["horizon"],
    oml_grace_period=fun_control["oml_grace_period"],
    metric=fun_control["metric_sklearn"],
)

```

```

from spotRiver.evaluation.eval_bml import plot_bml_oml_horizon_metrics, plot_bml_oml_horizon_predictions
df_labels=["default"]
plot_bml_oml_horizon_metrics(df_eval = [df_eval_default], log_y=False, df_labels=df_labels)
plot_bml_oml_horizon_predictions(df_true = [df_true_default[a:b]], target_column=target_column)

```



13.9.5 Get SPOT Results

```
from spotPython.hyperparameters.values import get_one_core_model_from_X
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
model_spot = get_one_core_model_from_X(X, fun_control)
model_spot
```

```
HoeffdingAdaptiveTreeRegressor (
  grace_period=657
  max_depth=256
  delta=4e-08
  tau=0.048426
  leaf_prediction="adaptive"
  leaf_model=LinearRegression (
    optimizer=SGD (
      lr=Constant (
        learning_rate=0.01
      )
    )
    loss=Squared ()
    l2=0.
    l1=0.
    intercept_init=0.
    intercept_lr=Constant (
      learning_rate=0.01
    )
    clip_gradient=1e+12
    initializer=Zeros ()
  )
  model_selector_decay=0.970713
  nominal_attributes=None
  splitter=QOSplitter (
    radius=0.25
    allow_multiway_splits=False
  )
  min_samples_split=5
  bootstrap_sampling=1
  drift_window_threshold=166
  drift_detector=ADWIN (
    delta=0.002
    clock=32
    max_buckets=5
  )
)
```

```

        min_window_length=5
        grace_period=10
    )
    switch_significance=0.040371
    binary_split=0
    max_size=454.140654
    memory_estimate_period=910594
    stop_mem_management=1
    remove_poor_attrs=1
    merit_preprune=0
    seed=None
)

```

```

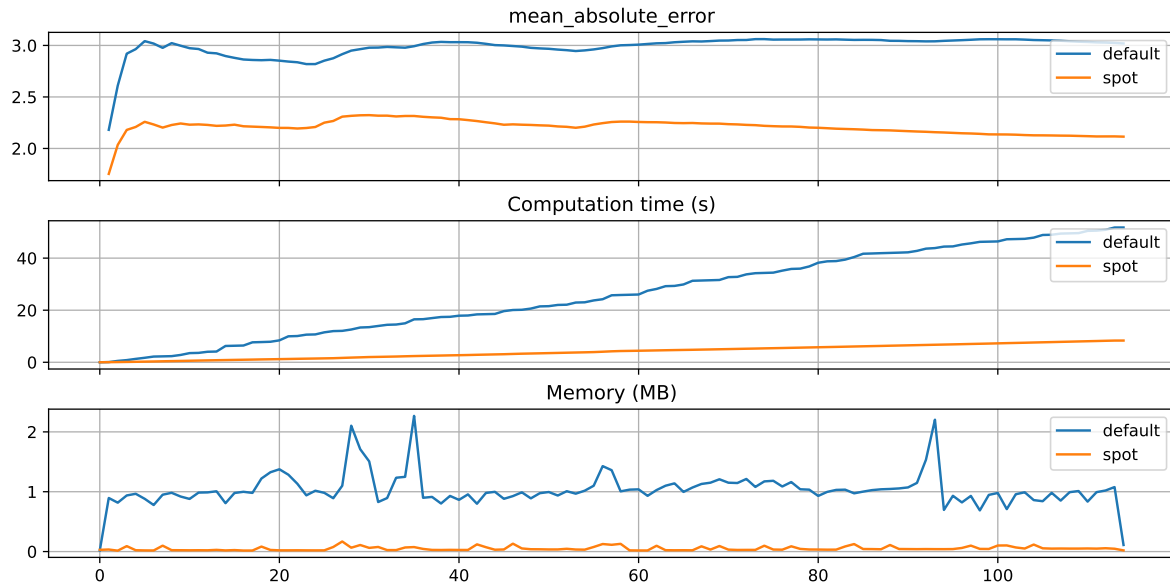
df_eval_spot, df_true_spot = eval_oml_horizon(
    model=model_spot,
    train=fun_control["train"],
    test=fun_control["test"],
    target_column=fun_control["target_column"],
    horizon=fun_control["horizon"],
    oml_grace_period=fun_control["oml_grace_period"],
    metric=fun_control["metric_sklearn"],
)

```

```

df_labels=["default", "spot"]
plot_bml_oml_horizon_metrics(df_eval = [df_eval_default, df_eval_spot], log_y=False, df_la

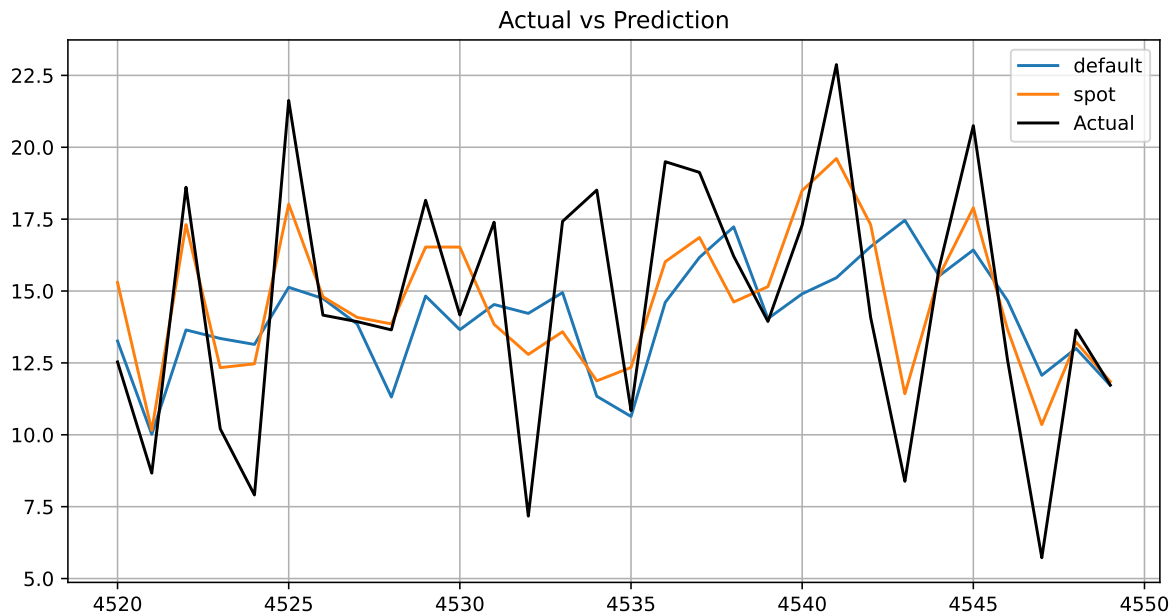
```



```

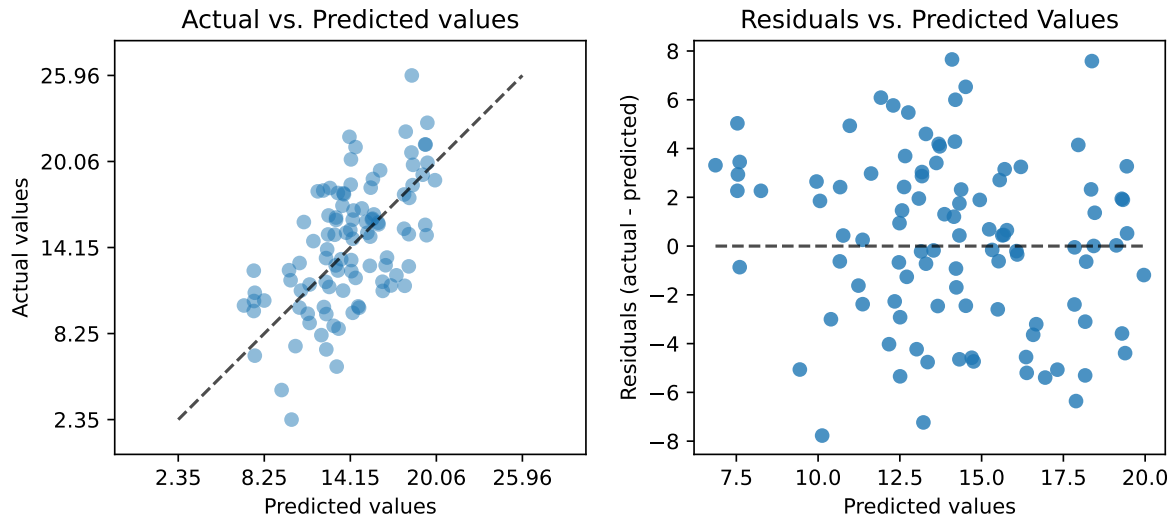
a = int(m/2)+20
b = int(m/2)+50
plot_bml_oml_horizon_predictions(df_true = [df_true_default[a:b], df_true_spot[a:b]], targ

```

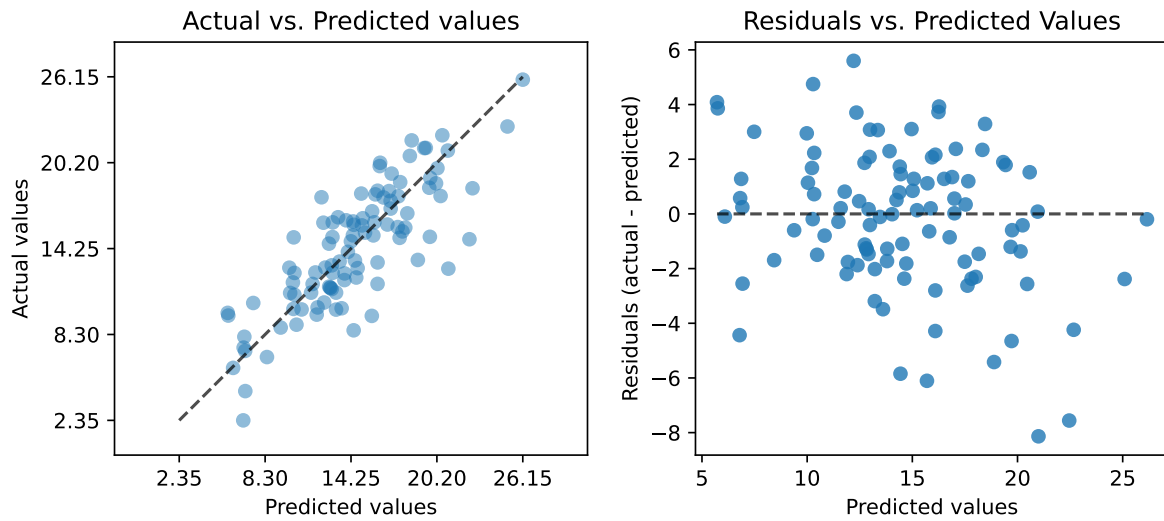


```
from spotPython.plot.validation import plot_actual_vs_predicted
plot_actual_vs_predicted(y_test=df_true_default["y"], y_pred=df_true_default["Prediction"])
plot_actual_vs_predicted(y_test=df_true_spot["y"], y_pred=df_true_spot["Prediction"], titl
```

Default



SPOT



13.9.6 Visualize Regression Trees

```
dataset_f = dataset.take(n_total)
for x, y in dataset_f:
    model_default.learn_one(x, y)
```

Caution: Large Trees

- Since the trees are large, the visualization is suppressed by default.
- To visualize the trees, uncomment the following line.

```
# model_default.draw()
```

```
model_default.summary
```

```
{'n_nodes': 35,
 'n_branches': 17,
 'n_leaves': 18,
 'n_active_leaves': 96,
 'n_inactive_leaves': 0,
 'height': 6,
 'total_observed_weight': 39002.0,
 'n_alternate_trees': 21,
 'n_pruned_alternate_trees': 6,
 'n_switch_alternate_trees': 2}
```

13.9.7 Spot Model

```
dataset_f = dataset.take(n_total)
for x, y in dataset_f:
    model_spot.learn_one(x, y)
```

Caution: Large Trees

- Since the trees are large, the visualization is suppressed by default.
- To visualize the trees, uncomment the following line.

```
# model_spot.draw()
```

```
model_spot.summary
```

```
{'n_nodes': 27,  
 'n_branches': 13,  
 'n_leaves': 14,  
 'n_active_leaves': 6,  
 'n_inactive_leaves': 0,  
 'height': 10,  
 'total_observed_weight': 39002.0,  
 'n_alternate_trees': 34,  
 'n_pruned_alternate_trees': 26,  
 'n_switch_alternate_trees': 2}
```

```
from spotPython.utils.eda import compare_two_tree_models  
print(compare_two_tree_models(model_default, model_spot))
```

Parameter	Default	Spot
n_nodes	35	27
n_branches	17	13
n_leaves	18	14
n_active_leaves	96	6
n_inactive_leaves	0	0
height	6	10
total_observed_weight	39002	39002
n_alternate_trees	21	34
n_pruned_alternate_trees	6	26
n_switch_alternate_trees	2	2

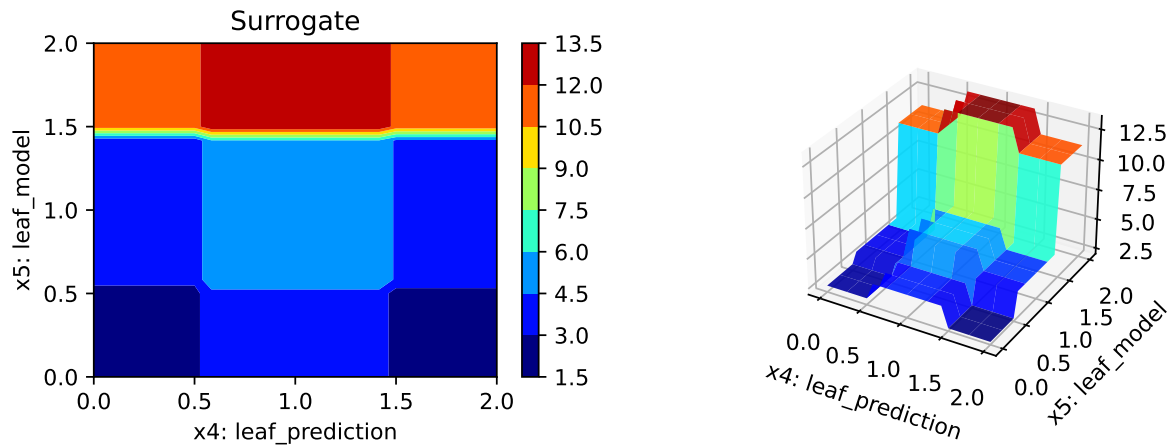
```
min(spot_tuner.y), max(spot_tuner.y)
```

```
(2.2211924973008017, 13.363463002223545)
```

13.9.8 Detailed Hyperparameter Plots

```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

```
leaf_prediction: 15.249473821718697
leaf_model: 100.0
```



13.9.9 Parallel Coordinates Plots

```
spot_tuner.parallel_plot()
```

Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): text/html

13.9.10 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

```
PLOT_ALL = False
if PLOT_ALL:
    n = spot_tuner.k
```

```
for i in range(n-1):  
    for j in range(i+1, n):  
        spot_tuner.plot_contour(i=i, j=j, min_z=min_z, max_z = max_z)
```


14 HPT: PyTorch With spotPython and Ray Tune on CIFAR10

In this tutorial, we will show how `spotPython` can be integrated into the `PyTorch` training workflow. It is based on the tutorial “Hyperparameter Tuning with Ray Tune” from the `PyTorch` documentation (PyTorch 2023a), which is an extension of the tutorial “Training a Classifier” (PyTorch 2023b) for training a CIFAR10 image classifier.

This document refers to the following software versions:

- `python`: 3.10.10
- `torch`: 2.0.1
- `torchvision`: 0.15.0

```
pip list | grep "spot[RiverPython]"
```

<code>spotPython</code>	0.2.52
-------------------------	--------

<code>spotRiver</code>	0.0.94
------------------------	--------

Note: you may need to restart the kernel to use updated packages.

`spotPython` can be installed via `pip`¹.

```
!pip install spotPython
```

- Uncomment the following lines if you want to for (re-)installation the latest version of `spotPython` from `gitHub`.


¹Alternatively, the source code can be downloaded from `gitHub`: <https://github.com/sequential-parameter-optimization/spotPython>.

```
# import sys
# !{sys.executable} -m pip install --upgrade build
# !{sys.executable} -m pip install --upgrade --force-reinstall spotPython
```

Results that refer to the Ray Tune package are taken from https://PyTorch.org/tutorials/beginner/hyperparameter_tuning_tutorial.html².

14.1 Step 1: Setup

Before we consider the detailed experimental setup, we select the parameters that affect run time, initial design size and the device that is used.

 **Caution:** Run time and initial design size should be increased for real experiments

- MAX_TIME is set to one minute for demonstration purposes. For real experiments, this should be increased to at least 1 hour.
- INIT_SIZE is set to 5 for demonstration purposes. For real experiments, this should be increased to at least 10.

 **Note:** Device selection

- The device can be selected by setting the variable DEVICE.
- Since we are using a simple neural net, the setting "cpu" is preferred (on Mac).
- If you have a GPU, you can use "cuda:0" instead.
- If DEVICE is set to None, spotPython will automatically select the device.
 - This might result in "mps" on Macs, which is not the best choice for simple neural nets.

```
MAX_TIME = 10
INIT_SIZE = 5
DEVICE = "cpu" # "cuda:0"
```

```
from spotPython.utils.device import getDevice
DEVICE = getDevice(DEVICE)
print(DEVICE)
```

cpu

²We were not able to install Ray Tune on our system. Therefore, we used the results from the PyTorch tutorial.

```

import os
import copy
import socket
import warnings
from datetime import datetime
from dateutil.tz import tzlocal
start_time = datetime.now(tzlocal())
HOSTNAME = socket.gethostname().split(".")[0]
experiment_name = '14-torch' + "_" + HOSTNAME + "_" + str(MAX_TIME) + "min_" + str(INIT_SECONDS)
experiment_name = experiment_name.replace(':', '-')
print(experiment_name)
if not os.path.exists('./figures'):
    os.makedirs('./figures')
warnings.filterwarnings("ignore")

```

14-torch_maans03_10min_5init_2023-07-03_12-18-15

14.2 Step 2: Initialization of the `fun_control` Dictionary

`spotPython` uses a Python dictionary for storing the information required for the hyperparameter tuning process. This dictionary is called `fun_control` and is initialized with the function `fun_control_init`. The function `fun_control_init` returns a skeleton dictionary. The dictionary is filled with the required information for the hyperparameter tuning process. It stores the hyperparameter tuning settings, e.g., the deep learning network architecture that should be tuned, the classification (or regression) problem, and the data that is used for the tuning. The dictionary is used as an input for the SPOT function.

 **Caution:** Tensorboard does not work under Windows

- Since tensorboard does not work under Windows, we recommend setting the parameter `tensorboard_path` to `None` if you are working under Windows.

```

from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="classification",
    tensorboard_path="runs/14_spot_ray_hpt_torch_cifar10",
    device=DEVICE,)

```

14.3 Step 3: PyTorch Data Loading

The data loading process is implemented in the same manner as described in the Section “Data loaders” in PyTorch (2023a). The data loaders are wrapped into the function `load_data_cifar10` which is identical to the function `load_data` in PyTorch (2023a). A global data directory is used, which allows sharing the data directory between different trials. The method `load_data_cifar10` is part of the `spotPython` package and can be imported from `spotPython.data.torchdata`.

In the following step, the test and train data are added to the dictionary `fun_control`.

```
from spotPython.data.torchdata import load_data_cifar10
train, test = load_data_cifar10()
n_samples = len(train)
# add the dataset to the fun_control
fun_control.update({
    "train": train,
    "test": test,
    "n_samples": n_samples})
```

Files already downloaded and verified

Files already downloaded and verified

14.4 Step 4: Specification of the Preprocessing Model

After the training and test data are specified and added to the `fun_control` dictionary, `spotPython` allows the specification of a data preprocessing pipeline, e.g., for the scaling of the data or for the one-hot encoding of categorical variables. The preprocessing model is called `prep_model` (“preparation” or pre-processing) and includes steps that are not subject to the hyperparameter tuning process. The preprocessing model is specified in the `fun_control` dictionary. The preprocessing model can be implemented as a `sklearn` pipeline. The following code shows a typical preprocessing pipeline:

```
categorical_columns = ["cities", "colors"]
one_hot_encoder = OneHotEncoder(handle_unknown="ignore",
                                sparse_output=False)
prep_model = ColumnTransformer(
    transformers=[
        ("categorical", one_hot_encoder, categorical_columns),
```

```

        ],
        remainder=StandardScaler(),
    )

```

Because the Ray Tune (`ray[tune]`) hyperparameter tuning as described in PyTorch (2023a) does not use a preprocessing model, the preprocessing model is set to `None` here.

```

prep_model = None
fun_control.update({"prep_model": prep_model})

```

14.5 Step 5: Select Model (algorithm) and `core_model_hyper_dict`

The same neural network model as implemented in the section “Configurable neural network” of the PyTorch tutorial (PyTorch 2023a) is used here. We will show the implementation from PyTorch (2023a) in Section 14.5.0.1 first, before the extended implementation with `spotPython` is shown in Section 14.5.0.2.

14.5.0.1 Implementing a Configurable Neural Network With Ray Tune

We used the same hyperparameters that are implemented as configurable in the PyTorch tutorial. We specify the layer sizes, namely 11 and 12, of the fully connected layers:

```

class Net(nn.Module):
    def __init__(self, l1=120, l2=84):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, l1)
        self.fc2 = nn.Linear(l1, l2)
        self.fc3 = nn.Linear(l2, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))

```

```

        x = self.fc3(x)
    return x

```

The learning rate, i.e., `lr`, of the optimizer is made configurable, too:

```
optimizer = optim.SGD(net.parameters(), lr=config["lr"], momentum=0.9)
```

14.5.0.2 Implementing a Configurable Neural Network With `spotPython`

`spotPython` implements a class which is similar to the class described in the PyTorch tutorial. The class is called `Net_CIFAR10` and is implemented in the file `netcifar10.py`.

```

from torch import nn
import torch.nn.functional as F
import spotPython.torch.netcore as netcore

class Net_CIFAR10(netcore.Net_Core):
    def __init__(self, l1, l2, lr_mult, batch_size, epochs, k_folds, patience,
optimizer, sgd_momentum):
        super(Net_CIFAR10, self).__init__(
            lr_mult=lr_mult,
            batch_size=batch_size,
            epochs=epochs,
            k_folds=k_folds,
            patience=patience,
            optimizer=optimizer,
            sgd_momentum=sgd_momentum,
        )
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 11)
        self.fc2 = nn.Linear(11, 12)
        self.fc3 = nn.Linear(12, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))

```

```

        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

```

14.5.1 The Net_Core class

`Net_CIFAR10` inherits from the class `Net_Core` which is implemented in the file `netcore.py`. It implements the additional attributes that are common to all neural network models. The `Net_Core` class is implemented in the file `netcore.py`. It implements hyperparameters as attributes, that are not used by the `core_model`, e.g.:

- optimizer (`optimizer`),
- learning rate (`lr`),
- batch size (`batch_size`),
- epochs (`epochs`),
- k_folds (`k_folds`), and
- early stopping criterion “patience” (`patience`).

Users can add further attributes to the class. The class `Net_Core` is shown below.

```

from torch import nn

class Net_Core(nn.Module):
    def __init__(self, lr_mult, batch_size, epochs, k_folds, patience,
                 optimizer, sgd_momentum):
        super(Net_Core, self).__init__()
        self.lr_mult = lr_mult
        self.batch_size = batch_size
        self.epochs = epochs
        self.k_folds = k_folds
        self.patience = patience
        self.optimizer = optimizer
        self.sgd_momentum = sgd_momentum

```

14.5.2 Comparison of the Approach Described in the PyTorch Tutorial With spotPython

Comparing the class `Net` from the PyTorch tutorial and the class `Net_CIFAR10` from `spotPython`, we see that the class `Net_CIFAR10` has additional attributes and does not inherit from `nn` directly. It adds an additional class, `Net_core`, that takes care of additional

attributes that are common to all neural network models, e.g., the learning rate multiplier `lr_mult` or the batch size `batch_size`.

`spotPython`'s `core_model` implements an instance of the `Net_CIFAR10` class. In addition to the basic neural network model, the `core_model` can use these additional attributes. `spotPython` provides methods for handling these additional attributes to guarantee 100% compatibility with the PyTorch classes. The method `add_core_model_to_fun_control` adds the hyperparameters and additional attributes to the `fun_control` dictionary. The method is shown below.

```
from spotPython.torch.netcifar10 import Net_CIFAR10
from spotPython.data.torch_hyper_dict import TorchHyperDict
from spotPython.hyperparameters.values import add_core_model_to_fun_control
core_model = Net_CIFAR10
fun_control = add_core_model_to_fun_control(core_model=core_model,
                                           fun_control=fun_control,
                                           hyper_dict=TorchHyperDict,
                                           filename=None)
```

14.5.3 The Search Space: Hyperparameters

In Section 14.5.4, we first describe how to configure the search space with `ray[tune]` (as shown in PyTorch (2023a)) and then how to configure the search space with `spotPython` in -14.

14.5.4 Configuring the Search Space With Ray Tune

Ray Tune's search space can be configured as follows (PyTorch 2023a):

```
config = {
    "l1": tune.sample_from(lambda _: 2**np.random.randint(2, 9)),
    "l2": tune.sample_from(lambda _: 2**np.random.randint(2, 9)),
    "lr": tune.loguniform(1e-4, 1e-1),
    "batch_size": tune.choice([2, 4, 8, 16])
}
```

The `tune.sample_from()` function enables the user to define sample methods to obtain hyperparameters. In this example, the `l1` and `l2` parameters should be powers of 2 between 4 and 256, so either 4, 8, 16, 32, 64, 128, or 256. The `lr` (learning rate) should be uniformly sampled between 0.0001 and 0.1. Lastly, the batch size is a choice between 2, 4, 8, and 16.

At each trial, `ray[tune]` will randomly sample a combination of parameters from these search spaces. It will then train a number of models in parallel and find the best performing one

among these. `ray[tune]` uses the `ASHAScheduler` which will terminate bad performing trials early.

14.5.5 Configuring the Search Space With `spotPython`

14.5.5.1 The `hyper_dict` Hyperparameters for the Selected Algorithm

`spotPython` uses JSON files for the specification of the hyperparameters. Users can specify their individual JSON files, or they can use the JSON files provided by `spotPython`. The JSON file for the `core_model` is called `torch_hyper_dict.json`.

In contrast to `ray[tune]`, `spotPython` can handle numerical, boolean, and categorical hyperparameters. They can be specified in the JSON file in a similar way as the numerical hyperparameters as shown below. Each entry in the JSON file represents one hyperparameter with the following structure: `type`, `default`, `transform`, `lower`, and `upper`.

```
"factor_hyperparameter": {
  "levels": ["A", "B", "C"],
  "type": "factor",
  "default": "B",
  "transform": "None",
  "core_model_parameter_type": "str",
  "lower": 0,
  "upper": 2},
```

The corresponding entries for the `core_model` class are shown below.

```
fun_control['core_model_hyper_dict']
```

```
{'l1': {'type': 'int',
  'default': 5,
  'transform': 'transform_power_2_int',
  'lower': 2,
  'upper': 9},
'l2': {'type': 'int',
  'default': 5,
  'transform': 'transform_power_2_int',
  'lower': 2,
  'upper': 9},
'lr_mult': {'type': 'float',
  'default': 1.0,
  'transform': 'None',
```

```

'lower': 0.1,
'upper': 10.0},
'batch_size': {'type': 'int',
'default': 4,
'transform': 'transform_power_2_int',
'lower': 1,
'upper': 4},
'epochs': {'type': 'int',
'default': 3,
'transform': 'transform_power_2_int',
'lower': 3,
'upper': 4},
'k_folds': {'type': 'int',
'default': 1,
'transform': 'None',
'lower': 1,
'upper': 1},
'patience': {'type': 'int',
'default': 5,
'transform': 'None',
'lower': 2,
'upper': 10},
'optimizer': {'levels': ['Adadelata',
'Adagrad',
'Adam',
'AdamW',
'SparseAdam',
'Adamax',
'ASGD',
'NAdam',
'RAdam',
'RMSprop',
'Rprop',
'SGD'],
'type': 'factor',
'default': 'SGD',
'transform': 'None',
'class_name': 'torch.optim',
'core_model_parameter_type': 'str',
'lower': 0,
'upper': 12},
'sgd_momentum': {'type': 'float',
'default': 0.0,

```

```
'transform': 'None',  
'lower': 0.0,  
'upper': 1.0}}
```

14.6 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model`

Ray tune (PyTorch 2023a) does not provide a way to change the specified hyperparameters without re-compilation. However, `spotPython` provides functions for modifying the hyperparameters, their bounds and factors as well as for activating and de-activating hyperparameters without re-compilation of the Python source code. These functions are described in the following.

14.6.0.1 Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model`

After specifying the model, the corresponding hyperparameters, their types and bounds are loaded from the JSON file `torch_hyper_dict.json`. After loading, the user can modify the hyperparameters, e.g., the bounds. `spotPython` provides a simple rule for de-activating hyperparameters: If the lower and the upper bound are set to identical values, the hyperparameter is de-activated. This is useful for the hyperparameter tuning, because it allows to specify a hyperparameter in the JSON file, but to de-activate it in the `fun_control` dictionary. This is done in the next step.

14.6.0.2 Modify Hyperparameters of Type numeric and integer (boolean)

Since the hyperparameter `k_folds` is not used in the PyTorch tutorial, it is de-activated here by setting the lower and upper bound to the same value. Note, `k_folds` is of type “integer”.

```
from spotPython.hyperparameters.values import modify_hyper_parameter_bounds  
fun_control = modify_hyper_parameter_bounds(fun_control,  
                                             "batch_size", bounds=[1, 5])  
fun_control = modify_hyper_parameter_bounds(fun_control,  
                                             "k_folds", bounds=[0, 0])  
fun_control = modify_hyper_parameter_bounds(fun_control,  
                                             "patience", bounds=[3, 3])
```

14.6.0.3 Modify Hyperparameter of Type factor

In a similar manner as for the numerical hyperparameters, the categorical hyperparameters can be modified. New configurations can be chosen by adding or deleting levels. For example, the hyperparameter `optimizer` can be re-configured as follows:

In the following setting, two optimizers ("SGD" and "Adam") will be compared during the `spotPython` hyperparameter tuning. The hyperparameter `optimizer` is active.

```
from spotPython.hyperparameters.values import modify_hyper_parameter_levels
fun_control = modify_hyper_parameter_levels(fun_control,
                                           "optimizer", ["SGD", "Adam"])
```

The hyperparameter `optimizer` can be de-activated by choosing only one value (level), here: "SGD".

```
fun_control = modify_hyper_parameter_levels(fun_control, "optimizer", ["SGD"])
```

As discussed in Section 14.6.1, there are some issues with the LBFGS optimizer. Therefore, the usage of the LBFGS optimizer is not deactivated in `spotPython` by default. However, the LBFGS optimizer can be activated by adding it to the list of optimizers. `Rprop` was removed, because it does perform very poorly (as some pre-tests have shown). However, it can also be activated by adding it to the list of optimizers. Since `SparseAdam` does not support dense gradients, `Adam` was used instead. Therefore, there are 10 default optimizers:

```
fun_control = modify_hyper_parameter_levels(fun_control, "optimizer",
                                           ["Adadelta", "Adagrad", "Adam", "AdamW", "Adamax", "ASGD",
                                           "NAdam", "RAdam", "RMSprop", "SGD"])
```

14.6.1 Optimizers

Table 14.1 shows some of the optimizers available in PyTorch:

a denotes (0.9,0.999), b (0.5,1.2), and c (1e-6, 50), respectively. R denotes required, but unspecified. "m" denotes momentum, "w_d" weight_decay, "d" dampening, "n" nesterov, "r" rho, "l_s" learning rate for scaling delta, "l_d" lr_decay, "b" betas, "l" lambda, "a" alpha, "m_d" for momentum_decay, "e" etas, and "s_s" for step_sizes.

Table 14.1: Optimizers available in PyTorch (selection). The default values are shown in the table.

Optimizer	lr	m	w_d	d	n	r	l_s	l_d	b	l	a	m_d	e	s_s
Adadelta	-	-	0.	-	-	0.9	1.	-	-	-	-	-	-	-
Adagrad	1e-2	-	0.	-	-	-	-	0.	-	-	-	-	-	-
Adam	1e-3	-	0.	-	-	-	-	-	<i>a</i>	-	-	-	-	-
AdamW	1e-3	-	1e-2	-	-	-	-	-	<i>a</i>	-	-	-	-	-
SparseAdam	1e-3	-	-	-	-	-	-	-	<i>a</i>	-	-	-	-	-
Adamax	2e-3	-	0.	-	-	-	-	-	<i>a</i>	-	-	-	-	-
ASGD	1e-2	.9	0.	-	F	-	-	-	-	1e-4	.75	-	-	-
LBFGS	1.	-	-	-	-	-	-	-	-	-	-	-	-	-
NAdam	2e-3	-	0.	-	-	-	-	-	<i>a</i>	-	-	0	-	-
RAdam	1e-3	-	0.	-	-	-	-	-	<i>a</i>	-	-	-	-	-
RMSprop	1e-2	0.	0.	-	-	-	-	-	<i>a</i>	-	-	-	-	-
Rprop	1e-2	-	-	-	-	-	-	-	-	-	<i>b</i>	<i>c</i>	-	-
SGD	<i>R</i>	0.	0.	0.	F	-	-	-	-	-	-	-	-	-

`spotPython` implements an `optimization` handler that maps the optimizer names to the corresponding PyTorch optimizers.

i A note on LBFGS

We recommend deactivating PyTorch’s LBFGS optimizer, because it does not perform very well. The PyTorch documentation, see <https://pytorch.org/docs/stable/generated/torch.optim.LBFGS.html#torch.optim.LBFGS>, states:

This is a very memory intensive optimizer (it requires additional `param_bytes * (history_size + 1)` bytes). If it doesn’t fit in memory try reducing the history size, or use a different algorithm.

Furthermore, the LBFGS optimizer is not compatible with the PyTorch tutorial. The reason is that the LBFGS optimizer requires the `closure` function, which is not implemented in the PyTorch tutorial. Therefore, the LBFGS optimizer is recommended here. Since there are ten optimizers in the portfolio, it is not recommended tuning the hyperparameters that effect one single optimizer only.

i A note on the learning rate

`spotPython` provides a multiplier for the default learning rates, `lr_mult`, because optimizers use different learning rates. Using a multiplier for the learning rates might enable

a simultaneous tuning of the learning rates for all optimizers. However, this is not recommended, because the learning rates are not comparable across optimizers. Therefore, we recommend fixing the learning rate for all optimizers if multiple optimizers are used. This can be done by setting the lower and upper bounds of the learning rate multiplier to the same value as shown below.

Thus, the learning rate, which affects the SGD optimizer, will be set to a fixed value. We choose the default value of `1e-3` for the learning rate, because it is used in other PyTorch examples (it is also the default value used by `spotPython` as defined in the `optimizer_handler()` method). We recommend tuning the learning rate later, when a reduced set of optimizers is fixed. Here, we will demonstrate how to select in a screening phase the optimizers that should be used for the hyperparameter tuning.

For the same reason, we will fix the `sgd_momentum` to 0.9.

```
fun_control = modify_hyper_parameter_bounds(fun_control,
                                             "lr_mult", bounds=[1.0, 1.0])
fun_control = modify_hyper_parameter_bounds(fun_control,
                                             "sgd_momentum", bounds=[0.9, 0.9])
```

14.7 Step 7: Selection of the Objective (Loss) Function

14.7.1 Evaluation: Data Splitting

The evaluation procedure requires the specification of the way how the data is split into a train and a test set and the loss function (and a metric). As a default, `spotPython` provides a standard hold-out data split and cross validation.

14.7.2 Hold-out Data Split

If a hold-out data split is used, the data will be partitioned into a training, a validation, and a test data set. The split depends on the setting of the `eval` parameter. If `eval` is set to `train_hold_out`, one data set, usually the original training data set, is split into a new training and a validation data set. The training data set is used for training the model. The validation data set is used for the evaluation of the hyperparameter configuration and early stopping to prevent overfitting. In this case, the original test data set is not used.

i Note

`spotPython` returns the hyperparameters of the machine learning and deep learning models, e.g., number of layers, learning rate, or optimizer, but not the model weights. Therefore, after the SPOT run is finished, the corresponding model with the optimized architecture has to be trained again with the best hyperparameter configuration. The training is performed on the training data set. The test data set is used for the final evaluation of the model.

Summarizing, the following splits are performed in the hold-out setting:

1. Run `spotPython` with `eval` set to `train_hold_out` to determine the best hyperparameter configuration.
2. Train the model with the best hyperparameter configuration (“architecture”) on the training data set: `train_tuned(model_spot, train, "model_spot.pt")`.
3. Test the model on the test data: `test_tuned(model_spot, test, "model_spot.pt")`

These steps will be exemplified in the following sections.

In addition to this `hold-out` setting, `spotPython` provides another hold-out setting, where an explicit test data is specified by the user that will be used as the validation set. To choose this option, the `eval` parameter is set to `test_hold_out`. In this case, the training data set is used for the model training. Then, the explicitly defined test data set is used for the evaluation of the hyperparameter configuration (the validation).

14.7.3 Cross-Validation

The cross validation setting is used by setting the `eval` parameter to `train_cv` or `test_cv`. In both cases, the data set is split into k folds. The model is trained on $k - 1$ folds and evaluated on the remaining fold. This is repeated k times, so that each fold is used exactly once for evaluation. The final evaluation is performed on the test data set. The cross validation setting is useful for small data sets, because it allows to use all data for training and evaluation. However, it is computationally expensive, because the model has to be trained k times.

i Note

Combinations of the above settings are possible, e.g., cross validation can be used for training and hold-out for evaluation or *vice versa*. Also, cross validation can be used for training and testing. Because cross validation is not used in the PyTorch tutorial (PyTorch 2023a), it is not considered further here.

14.7.4 Overview of the Evaluation Settings

14.7.4.1 Settings for the Hyperparameter Tuning

An overview of the training evaluations is shown in Table 14.2. "train_cv" and "test_cv" use `sklearn.model_selection.KFold()` internally. More details on the data splitting are provided in Section 23.14 (in the Appendix).

Table 14.2: Overview of the evaluation settings.

eval	train	test	function	comment
"train_hold_out" ✓			<code>train_one_epoch()</code> , <code>validate_one_epoch()</code> for early stopping	splits the train data set internally
"test_hold_out" ✓	✓	✓	<code>train_one_epoch()</code> , <code>validate_one_epoch()</code> for early stopping	use the test data set for
"train_cv"	✓		<code>evaluate_cv(net, train)</code>	<code>validate_one_epoch()</code> CV using the train data set
"test_cv"		✓	<code>evaluate_cv(net, test)</code>	CV using the test data set . Identical to "train_cv", uses only test data.

14.7.4.2 Settings for the Final Evaluation of the Tuned Architecture

14.7.4.2.1 Training of the Tuned Architecture

`train_tuned(model, train)`: train the model with the best hyperparameter configuration (or simply the default) on the training data set. It splits the `traindata` into new `train` and `validation` sets using `create_train_val_data_loaders()`, which calls `torch.utils.data.random_split()` internally. Currently, 60% of the data is used for training and 40% for validation. The `train` data is used for training the model with `train_hold_out()`. The `validation` data is used for early stopping using `validate_fold_or_hold_out()` on the `validation` data set.

14.7.4.2.2 Testing of the Tuned Architecture

`test_tuned(model, test)`: test the model on the test data set. No data splitting is performed. The (trained) model is evaluated using the `validate_fold_or_hold_out()` function. Note: During training, "shuffle" is set to `True`, whereas during testing, "shuffle" is set to `False`.

Section 23.14.1.4 describes the final evaluation of the tuned architecture.

```
fun_control.update({
    "eval": "train_hold_out",
    "path": "torch_model.pt",
    "shuffle": True})
```

14.7.5 Evaluation: Loss Functions and Metrics

The key "loss_function" specifies the loss function which is used during the optimization. There are several different loss functions under PyTorch's `nn` package. For example, a simple loss is `MSELoss`, which computes the mean-squared error between the output and the target. In this tutorial we will use `CrossEntropyLoss`, because it is also used in the PyTorch tutorial.

```
from torch.nn import CrossEntropyLoss
loss_function = CrossEntropyLoss()
fun_control.update({"loss_function": loss_function})
```

In addition to the loss functions, `spotPython` provides access to a large number of metrics.

- The key "metric_sklearn" is used for metrics that follow the `scikit-learn` conventions.
- The key "river_metric" is used for the river based evaluation (Montiel et al. 2021) via `eval_oml_iter_progressive`, and
- the key "metric_torch" is used for the metrics from `TorchMetrics`.

`TorchMetrics` is a collection of more than 90 PyTorch metrics, see <https://torchmetrics.readthedocs.io/en/latest/>. Because the PyTorch tutorial uses the accuracy as metric, we use the same metric here. Currently, accuracy is computed in the tutorial's example code. We will use `TorchMetrics` instead, because it offers more flexibility, e.g., it can be used for regression and classification. Furthermore, `TorchMetrics` offers the following advantages:

- * A standardized interface to increase reproducibility
- * Reduces Boilerplate
- * Distributed-training compatible
- * Rigorously tested
- * Automatic accumulation over batches
- * Automatic synchronization between multiple devices

Therefore, we set

```
import torchmetrics
metric_torch = torchmetrics.Accuracy(task="multiclass", num_classes=10).to(fun_control["device"])
fun_control.update({"metric_torch": metric_torch})
```

14.8 Step 8: Calling the SPOT Function

14.8.1 Preparing the SPOT Call

The following code passes the information about the parameter ranges and bounds to `spot`.

```
from spotPython.hyperparameters.values import (
    get_var_type,
    get_var_name,
    get_bound_values
)
var_type = get_var_type(fun_control)
var_name = get_var_name(fun_control)
fun_control.update({"var_type": var_type,
                   "var_name": var_name})

lower = get_bound_values(fun_control, "lower")
upper = get_bound_values(fun_control, "upper")
```

Now, the dictionary `fun_control` contains all information needed for the hyperparameter tuning. Before the hyperparameter tuning is started, it is recommended to take a look at the experimental design. The method `gen_design_table` generates a design table as follows:

```
from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control))
```

name	type	default	lower	upper	transform
l1	int	5	2	9	transform_power_2_int
l2	int	5	2	9	transform_power_2_int
lr_mult	float	1.0	1	1	None
batch_size	int	4	1	5	transform_power_2_int
epochs	int	3	3	4	transform_power_2_int
k_folds	int	1	0	0	None
patience	int	5	3	3	None
optimizer	factor	SGD	0	9	None

	sgd_momentum		float		0.0		0.9		0.9		None	
--	--------------	--	-------	--	-----	--	-----	--	-----	--	------	--

This allows to check if all information is available and if the information is correct. **?@tbl-design** shows the experimental design for the hyperparameter tuning. The table shows the hyperparameters, their types, default values, lower and upper bounds, and the transformation function. The transformation function is used to transform the hyperparameter values from the unit hypercube to the original domain. The transformation function is applied to the hyperparameter values before the evaluation of the objective function. Hyperparameter transformations are shown in the column “transform”, e.g., the `l1` default is 5, which results in the value $2^5 = 32$ for the network, because the transformation `transform_power_2_int` was selected in the JSON file. The default value of the `batch_size` is set to 4, which results in a batch size of $2^4 = 16$.

14.8.2 The Objective Function `fun_torch`

The objective function `fun_torch` is selected next. It implements an interface from PyTorch’s training, validation, and testing methods to `spotPython`.

```
from spotPython.fun.hypertorch import HyperTorch
fun = HyperTorch().fun_torch
```

14.8.3 Using Default Hyperparameters or Results from Previous Runs

We add the default setting to the initial design:

```
from spotPython.hyperparameters.values import get_default_hyperparameters_as_array
hyper_dict=TorchHyperDict().load()
X_start = get_default_hyperparameters_as_array(fun_control, hyper_dict)
```

14.8.4 Starting the Hyperparameter Tuning

The `spotPython` hyperparameter tuning is started by calling the `Spot` function. Here, we will run the tuner for approximately 30 minutes (`max_time`). Note: the initial design is always evaluated in the `spotPython` run. As a consequence, the run may take longer than specified by `max_time`, because the evaluation time of initial design (here: `init_size`, 10 points) is performed independently of `max_time`. During the run, results from the training is shown. These results can be visualized with Tensorboard as will be shown in Section 14.9.

MulticlassAccuracy: 0.4939999878406525 | Loss: 1.3872022548675538 | Acc: 0.4940000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5164999961853027 | Loss: 1.3260287700653077 | Acc: 0.5165000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5462999939918518 | Loss: 1.2660414114952088 | Acc: 0.5463000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5632500052452087 | Loss: 1.2166201056480408 | Acc: 0.5632500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5610499978065491 | Loss: 1.2368356624603272 | Acc: 0.5610500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5813000202178955 | Loss: 1.1746846567153930 | Acc: 0.5813000000000000.
Epoch: 9 |

MulticlassAccuracy: 0.5954499840736389 | Loss: 1.1482583248138427 | Acc: 0.5954500000000000.
Epoch: 10 |

MulticlassAccuracy: 0.6000999808311462 | Loss: 1.1498764616966248 | Acc: 0.6001000000000000.
Epoch: 11 |

MulticlassAccuracy: 0.5925999879837036 | Loss: 1.1826003037452697 | Acc: 0.5926000000000000.
Epoch: 12 |

MulticlassAccuracy: 0.5954999923706055 | Loss: 1.1827107128143310 | Acc: 0.5955000000000000.
Early stopping at epoch 11
Returned to Spot: Validation loss: 1.182710712814331

config: {'l1': 16, 'l2': 16, 'lr_mult': 1.0, 'batch_size': 8, 'epochs': 8, 'k_folds': 0, 'pa
Epoch: 1 |

MulticlassAccuracy: 0.4435999989509583 | Loss: 1.5066036984682083 | Acc: 0.4436000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.4920499920845032 | Loss: 1.3805544215440750 | Acc: 0.4920500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.5090500116348267 | Loss: 1.3838436133325100 | Acc: 0.5090500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5134000182151794 | Loss: 1.3652919138789177 | Acc: 0.5134000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5425000190734863 | Loss: 1.2813396899223328 | Acc: 0.5425000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5339499711990356 | Loss: 1.3223103511393071 | Acc: 0.5339500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5522000193595886 | Loss: 1.2933707284569740 | Acc: 0.5522000000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5381000041961670 | Loss: 1.3226890068054200 | Acc: 0.5381000000000000.
Early stopping at epoch 7
Returned to Spot: Validation loss: 1.32268900680542

config: {'l1': 256, 'l2': 128, 'lr_mult': 1.0, 'batch_size': 2, 'epochs': 16, 'k_folds': 0,
Epoch: 1 |

MulticlassAccuracy: 0.0981499999761581 | Loss: 2.3106271007537842 | Acc: 0.0981500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.1515000015497208 | Loss: 2.3217867916107178 | Acc: 0.1515000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.0960500016808510 | Loss: 2.3049573634743692 | Acc: 0.0960500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.1013500019907951 | Loss: 2.3090307576417923 | Acc: 0.1013500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.0958999991416931 | Loss: 2.3108412470340727 | Acc: 0.0959000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5160499811172485 | Loss: 1.3293344084262848 | Acc: 0.5160500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5411000251770020 | Loss: 1.2745791875362396 | Acc: 0.5411000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5455999970436096 | Loss: 1.2595453467369080 | Acc: 0.5456000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5518000125885010 | Loss: 1.2371071474075317 | Acc: 0.5518000000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5559499859809875 | Loss: 1.2294477246284485 | Acc: 0.5559500000000001.
Epoch: 8 |

MulticlassAccuracy: 0.5568500161170959 | Loss: 1.2272820011138916 | Acc: 0.5568500000000000.
Epoch: 9 |

MulticlassAccuracy: 0.5687500238418579 | Loss: 1.2133782124042511 | Acc: 0.5687500000000000.
Epoch: 10 |

MulticlassAccuracy: 0.5718500018119812 | Loss: 1.1942334382772446 | Acc: 0.5718500000000000.
Epoch: 11 |

MulticlassAccuracy: 0.5691499710083008 | Loss: 1.1962762669086455 | Acc: 0.5691500000000000.
Epoch: 12 |

MulticlassAccuracy: 0.5765500068664551 | Loss: 1.1818930788278579 | Acc: 0.5765500000000000.
Epoch: 13 |

MulticlassAccuracy: 0.5777999758720398 | Loss: 1.1739344074010849 | Acc: 0.5778000000000000.
Epoch: 14 |

MulticlassAccuracy: 0.5824999809265137 | Loss: 1.1722334628105164 | Acc: 0.5825000000000000.
Epoch: 15 |

MulticlassAccuracy: 0.5823000073432922 | Loss: 1.1642311267137528 | Acc: 0.5823000000000000.
Epoch: 16 |

MulticlassAccuracy: 0.5831000208854675 | Loss: 1.1653465260744096 | Acc: 0.5831000000000000.
Returned to Spot: Validation loss: 1.1653465260744096

config: {'l1': 64, 'l2': 256, 'lr_mult': 1.0, 'batch_size': 16, 'epochs': 16, 'k_folds': 0,
Epoch: 1 |

MulticlassAccuracy: 0.4367499947547913 | Loss: 1.5347009962558746 | Acc: 0.4367500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.4698500037193298 | Loss: 1.4526902680397034 | Acc: 0.4698500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.4846000075340271 | Loss: 1.4155467565536499 | Acc: 0.4846000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.4973999857902527 | Loss: 1.3829295865058899 | Acc: 0.4974000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5073999762535095 | Loss: 1.3607472479343414 | Acc: 0.5074000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5196999907493591 | Loss: 1.3298574501514435 | Acc: 0.5197000000000001.
Epoch: 7 |

MulticlassAccuracy: 0.5314999818801880 | Loss: 1.3078917122840881 | Acc: 0.5315000000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5324500203132629 | Loss: 1.3053160723209380 | Acc: 0.5324500000000000.
Epoch: 9 |

MulticlassAccuracy: 0.5345500111579895 | Loss: 1.2932133648872375 | Acc: 0.5345500000000000.
Epoch: 10 |

MulticlassAccuracy: 0.5436000227928162 | Loss: 1.2697099392890929 | Acc: 0.5436000000000000.
Epoch: 11 |

MulticlassAccuracy: 0.5429499745368958 | Loss: 1.2708870210170746 | Acc: 0.5429500000000000.
Epoch: 12 |

MulticlassAccuracy: 0.5517500042915344 | Loss: 1.2512038480281831 | Acc: 0.5517500000000000.
Epoch: 13 |

MulticlassAccuracy: 0.5482500195503235 | Loss: 1.2566084330558778 | Acc: 0.5482500000000000.
Epoch: 14 |

MulticlassAccuracy: 0.5580499768257141 | Loss: 1.2443617900848389 | Acc: 0.5580500000000000.
Epoch: 15 |

MulticlassAccuracy: 0.5563499927520752 | Loss: 1.2389848153114318 | Acc: 0.5563500000000000.
Epoch: 16 |

MulticlassAccuracy: 0.5600500106811523 | Loss: 1.2298900278329850 | Acc: 0.5600500000000000.
Returned to Spot: Validation loss: 1.229890027832985

spotPython tuning: 1.1653465260744096 [####-----] 35.92%

config: {'l1': 128, 'l2': 64, 'lr_mult': 1.0, 'batch_size': 32, 'epochs': 16, 'k_folds': 0,
Epoch: 1 |

MulticlassAccuracy: 0.4414499998092651 | Loss: 1.5097254679679870 | Acc: 0.4414500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.5005499720573425 | Loss: 1.3655176716804505 | Acc: 0.5005500000000001.
Epoch: 3 |

MulticlassAccuracy: 0.5336999893188477 | Loss: 1.2787800833702088 | Acc: 0.5337000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5530999898910522 | Loss: 1.2513076989173890 | Acc: 0.5531000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5716500282287598 | Loss: 1.1837824179649352 | Acc: 0.5716500000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5983499884605408 | Loss: 1.1271832468032836 | Acc: 0.5983500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5827500224113464 | Loss: 1.1788005928039551 | Acc: 0.5827500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5946000218391418 | Loss: 1.1742735905647277 | Acc: 0.5946000000000000.
Epoch: 9 |

MulticlassAccuracy: 0.6100000143051147 | Loss: 1.1143370146751403 | Acc: 0.6100000000000000.
Epoch: 10 |

MulticlassAccuracy: 0.6213499903678894 | Loss: 1.1004798809051515 | Acc: 0.6213500000000000.
Epoch: 11 |

MulticlassAccuracy: 0.6097999811172485 | Loss: 1.1298187858581543 | Acc: 0.6098000000000000.
Epoch: 12 |

MulticlassAccuracy: 0.6251000165939331 | Loss: 1.1078383001804353 | Acc: 0.6251000000000000.
Epoch: 13 |

MulticlassAccuracy: 0.6301500201225281 | Loss: 1.1029071352005004 | Acc: 0.6301500000000000.
Early stopping at epoch 12
Returned to Spot: Validation loss: 1.1029071352005004

spotPython tuning: 1.1029071352005004 [#####----] 61.83%

config: {'l1': 128, 'l2': 512, 'lr_mult': 1.0, 'batch_size': 32, 'epochs': 16, 'k_folds': 0,
Epoch: 1 |

MulticlassAccuracy: 0.4770500063896179 | Loss: 1.4390149123191833 | Acc: 0.4770500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.5321499705314636 | Loss: 1.3068075354576112 | Acc: 0.5321500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.5607500076293945 | Loss: 1.2343128045082092 | Acc: 0.5607500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5713000297546387 | Loss: 1.2181520700454711 | Acc: 0.5713000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.6044999957084656 | Loss: 1.1327894499778748 | Acc: 0.6045000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.6054000258445740 | Loss: 1.1293966197967529 | Acc: 0.6054000000000000.
Epoch: 7 |

MulticlassAccuracy: 0.6141999959945679 | Loss: 1.1064595009803773 | Acc: 0.6142000000000000.
Epoch: 8 |

MulticlassAccuracy: 0.6161500215530396 | Loss: 1.1265255500793456 | Acc: 0.6161500000000000.
Epoch: 9 |

MulticlassAccuracy: 0.6103000044822693 | Loss: 1.1644302755832672 | Acc: 0.6103000000000000.
Epoch: 10 |

MulticlassAccuracy: 0.6125500202178955 | Loss: 1.1852258719921112 | Acc: 0.6125500000000000.
Early stopping at epoch 9
Returned to Spot: Validation loss: 1.1852258719921112

spotPython tuning: 1.1029071352005004 [#####--] 82.11%

config: {'l1': 128, 'l2': 128, 'lr_mult': 1.0, 'batch_size': 32, 'epochs': 16, 'k_folds': 0,
Epoch: 1 |

MulticlassAccuracy: 0.4485999941825867 | Loss: 1.5299455617904663 | Acc: 0.4486000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.4990000128746033 | Loss: 1.3892097097396852 | Acc: 0.4990000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.5450999736785889 | Loss: 1.2654036801338195 | Acc: 0.5451000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5626000165939331 | Loss: 1.2238172842979431 | Acc: 0.5626000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5759500265121460 | Loss: 1.1976863938331603 | Acc: 0.5759500000000000.
Epoch: 6 |

```
MulticlassAccuracy: 0.5945000052452087 | Loss: 1.1594608989715576 | Acc: 0.5945000000000000.  
Epoch: 7 |  
  
MulticlassAccuracy: 0.5829499959945679 | Loss: 1.2049226930618286 | Acc: 0.5829500000000000.  
Epoch: 8 |  
  
MulticlassAccuracy: 0.5950000286102295 | Loss: 1.1615549116134642 | Acc: 0.5950000000000000.  
Epoch: 9 |  
  
MulticlassAccuracy: 0.6095499992370605 | Loss: 1.1511459247589111 | Acc: 0.6095500000000000.  
Epoch: 10 |  
  
MulticlassAccuracy: 0.6107500195503235 | Loss: 1.1499798847198486 | Acc: 0.6107500000000000.  
Epoch: 11 |  
  
MulticlassAccuracy: 0.6051499843597412 | Loss: 1.1655514258384705 | Acc: 0.6051500000000000.  
Epoch: 12 |  
  
MulticlassAccuracy: 0.6012499928474426 | Loss: 1.2182928497314454 | Acc: 0.6012500000000000.  
Epoch: 13 |  
  
MulticlassAccuracy: 0.6016499996185303 | Loss: 1.2296336574554443 | Acc: 0.6016500000000000.  
Early stopping at epoch 12  
Returned to Spot: Validation loss: 1.2296336574554443  
  
spotPython tuning: 1.1029071352005004 [#####] 100.00% Done...  
  
<spotPython.spot.spot.Spot at 0x187902740>
```

14.9 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard.

14.9.1 Tensorboard: Start Tensorboard

Start TensorBoard through the command line to visualize data you logged. Specify the root log directory as used in `fun_control = fun_control_init(task="regression", tensorboard_path="runs/24_spot_torch_regression")` as the `tensorboard_path`. The argument `logdir` points to directory where TensorBoard will look to find event files that it can display. TensorBoard will recursively walk the directory structure rooted at `logdir`, looking for `.tfevents` files.

```
tensorboard --logdir=runs
```

Go to the URL it provides or to <http://localhost:6006/>. The following figures show some screenshots of Tensorboard.

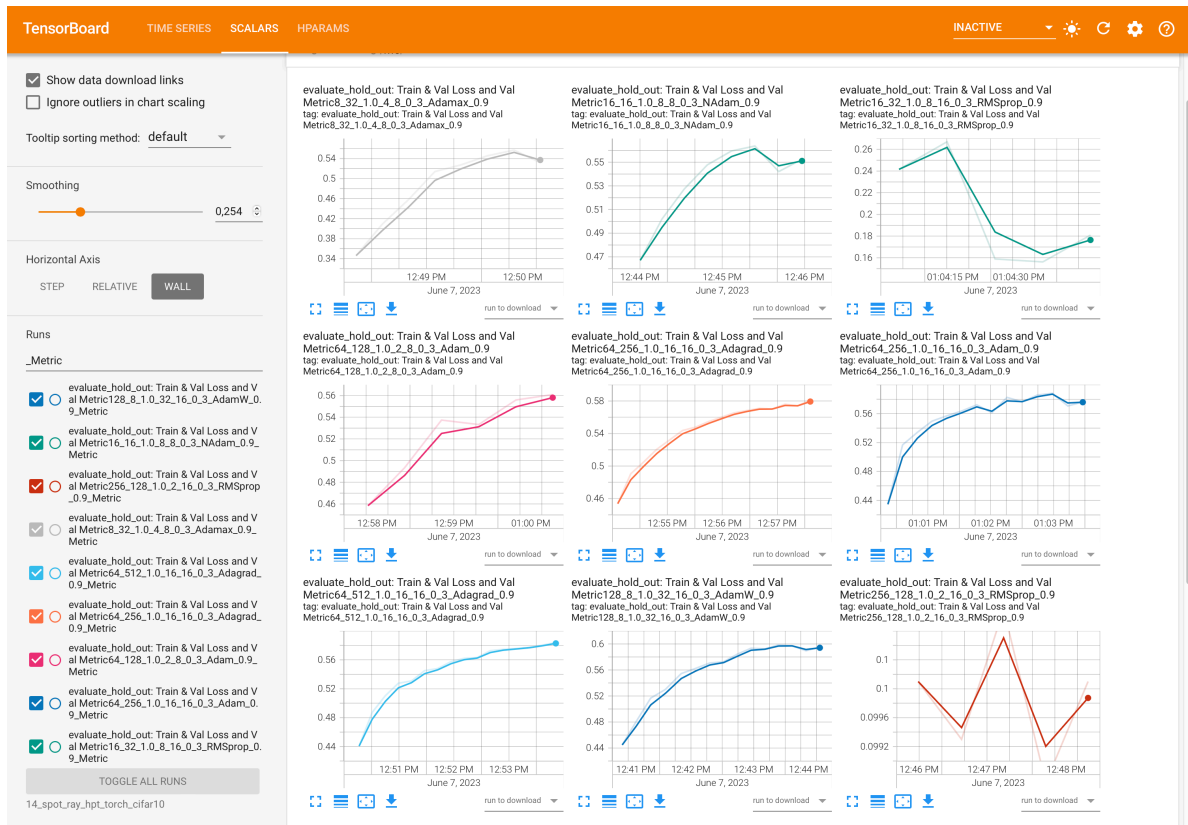


Figure 14.1: Tensorboard

Trial ID	Show Metrics	f1	f2	batch_size	epochs	patience	optimizer	fun_torch: loss
1686135261.24...	<input type="checkbox"/>	64.000	512.00	16.000	16.000	3.0000	Adagrad	1.1765
1686135486.0...	<input type="checkbox"/>	64.000	256.00	16.000	16.000	3.0000	Adagrad	1.1963
1686134673.15...	<input type="checkbox"/>	128.00	8.0000	32.000	16.000	3.0000	AdamW	1.2062
1686134773.50...	<input type="checkbox"/>	16.000	16.000	8.0000	8.0000	3.0000	NAdam	1.2880
1686135837.96...	<input type="checkbox"/>	64.000	256.00	16.000	16.000	3.0000	Adam	1.3155
1686135032.11...	<input type="checkbox"/>	8.0000	32.000	4.0000	8.0000	3.0000	Adamax	1.3435
1686135637.40...	<input type="checkbox"/>	64.000	128.00	2.0000	8.0000	3.0000	Adam	1.5804
1686135892.6...	<input type="checkbox"/>	16.000	32.000	8.0000	16.000	3.0000	RMSprop	2.1542
1686134917.07...	<input type="checkbox"/>	256.00	128.00	2.0000	16.000	3.0000	RMSprop	2.3099

Figure 14.2: Tensorboard

14.9.2 Saving the State of the Notebook

The state of the notebook can be saved and reloaded as follows:

```
import pickle
SAVE = False
LOAD = False

if SAVE:
    result_file_name = "res_" + experiment_name + ".pkl"
    with open(result_file_name, 'wb') as f:
        pickle.dump(spot_tuner, f)

if LOAD:
    result_file_name = "add_the_name_of_the_result_file_here.pkl"
    with open(result_file_name, 'rb') as f:
        spot_tuner = pickle.load(f)
```

14.10 Step 10: Results

After the hyperparameter tuning run is finished, the progress of the hyperparameter tuning can be visualized. The following code generates the progress plot from `?@fig-progress`.

```
spot_tuner.plot_progress(log_y=False,
    filename="./figures/" + experiment_name+"_progress.png")
```

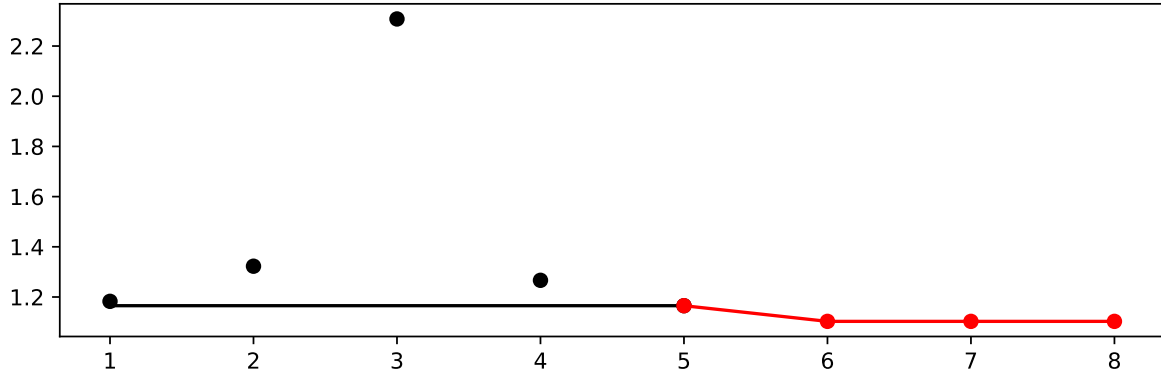


Figure 14.3: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

`?@fig-progress` shows a typical behaviour that can be observed in many hyperparameter studies (Bartz et al. 2022): the largest improvement is obtained during the evaluation of the initial design. The surrogate model based optimization refines the results. `?@fig-progress` also illustrates one major difference between `ray[tune]` as used in PyTorch (2023a) and `spotPython`: the `ray[tune]` uses a random search and will generate results similar to the *black* dots, whereas `spotPython` uses a surrogate model based optimization and presents results represented by *red* dots in `?@fig-progress`. The surrogate model based optimization is considered to be more efficient than a random search, because the surrogate model guides the search towards promising regions in the hyperparameter space.

In addition to the improved (“optimized”) hyperparameter values, `spotPython` allows a statistical analysis, e.g., a sensitivity analysis, of the results. We can print the results of the hyperparameter tuning, see `?@tbl-results`. The table shows the hyperparameters, their types, default values, lower and upper bounds, and the transformation function. The column “tuned” shows the tuned values. The column “importance” shows the importance of the hyperparameters. The column “stars” shows the importance of the hyperparameters in stars. The importance is computed by the SPOT software.

```
from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control=fun_control, spot=spot_tuner))
```

name	type	default	lower	upper	tuned	transform
l1	int	5	2.0	9.0	7.0	transform_power_2_int
l2	int	5	2.0	9.0	6.0	transform_power_2_int
lr_mult	float	1.0	1.0	1.0	1.0	None
batch_size	int	4	1.0	5.0	5.0	transform_power_2_int

epochs	int	3		3.0		4.0		4.0		transform_power_2_int	
k_folds	int	1		0.0		0.0		0.0		None	
patience	int	5		3.0		3.0		3.0		None	
optimizer	factor	SGD		0.0		9.0		3.0		None	
sgd_momentum	float	0.0		0.9		0.9		0.9		None	

To visualize the most important hyperparameters, `spotPython` provides the function `plot_importance`. The following code generates the importance plot from `?@fig-importance`.

```
spot_tuner.plot_importance(threshold=0.025,
                           filename="./figures/" + experiment_name+"_importance.png")
```

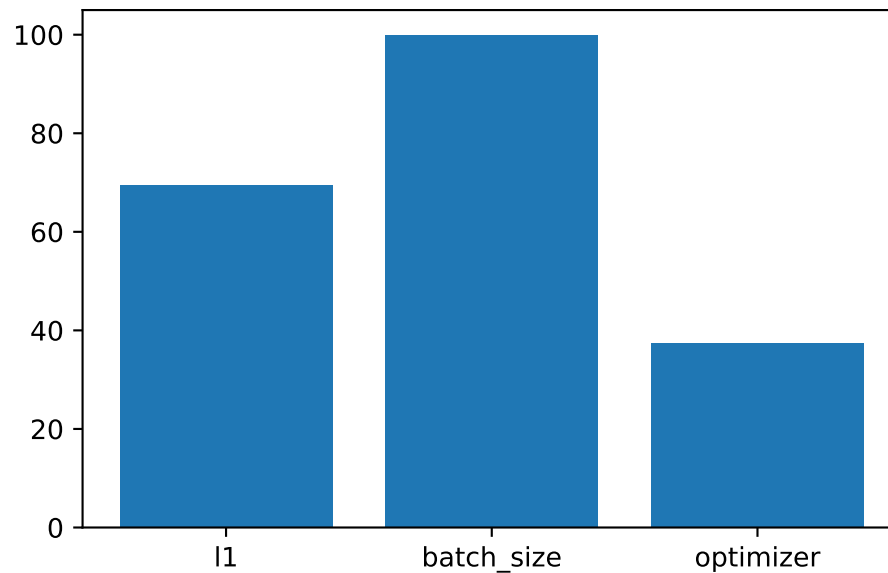


Figure 14.4: Variable importance plot, threshold 0.025.

14.10.1 Get the Tuned Architecture (SPOT Results)

The architecture of the `spotPython` model can be obtained as follows. First, the numerical representation of the hyperparameters are obtained, i.e., the numpy array `X` is generated. This array is then used to generate the model `model_spot` by the function `get_one_core_model_from_X`. The model `model_spot` has the following architecture:

```

from spotPython.hyperparameters.values import get_one_core_model_from_X
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
model_spot = get_one_core_model_from_X(X, fun_control)
model_spot

```

```

Net_CIFAR10(
    (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
    (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (fc1): Linear(in_features=400, out_features=128, bias=True)
    (fc2): Linear(in_features=128, out_features=64, bias=True)
    (fc3): Linear(in_features=64, out_features=10, bias=True)
)

```

14.10.2 Get Default Hyperparameters

In a similar manner as in Section 14.10.1, the default hyperparameters can be obtained.

```

# fun_control was modified, we generate a new one with the original
# default hyperparameters
from spotPython.hyperparameters.values import get_one_core_model_from_X
fc = fun_control
fc.update({"core_model_hyper_dict":
    hyper_dict[fun_control["core_model"].__name__]})
model_default = get_one_core_model_from_X(X_start, fun_control=fc)
model_default

```

```

Net_CIFAR10(
    (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
    (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (fc1): Linear(in_features=400, out_features=32, bias=True)
    (fc2): Linear(in_features=32, out_features=32, bias=True)
    (fc3): Linear(in_features=32, out_features=10, bias=True)
)

```

14.10.3 Evaluation of the Default Architecture

The method `train_tuned` takes a model architecture without trained weights and trains this model with the train data. The train data is split into train and validation data. The validation

data is used for early stopping. The trained model weights are saved as a dictionary. This evaluation is similar to the final evaluation in PyTorch (2023a).

```
from spotPython.torch.traintest import (
    train_tuned,
    test_tuned,
)
train_tuned(net=model_default, train_dataset=train, shuffle=True,
            loss_function=fun_control["loss_function"],
            metric=fun_control["metric_torch"],
            device = fun_control["device"], show_batch_interval=1_000_000,
            path=None,
            task=fun_control["task"],)

test_tuned(net=model_default, test_dataset=test,
            loss_function=fun_control["loss_function"],
            metric=fun_control["metric_torch"],
            shuffle=False,
            device = fun_control["device"],
            task=fun_control["task"],)
```

Epoch: 1 |

MulticlassAccuracy: 0.1009500026702881 | Loss: 2.3057962629318238 | Acc: 0.1009500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.1108499988913536 | Loss: 2.3033584295272829 | Acc: 0.1108500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.1353999972343445 | Loss: 2.3007775247573852 | Acc: 0.1354000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.1523499935865402 | Loss: 2.2973377077102661 | Acc: 0.1523500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.1611000001430511 | Loss: 2.2919936273574830 | Acc: 0.1611000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.1942500025033951 | Loss: 2.2823992488861085 | Acc: 0.1942500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.1798499971628189 | Loss: 2.2623666219711303 | Acc: 0.1798500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.1803999990224838 | Loss: 2.2218215888977051 | Acc: 0.1804000000000000.
Returned to Spot: Validation loss: 2.221821588897705

MulticlassAccuracy: 0.1836999952793121 | Loss: 2.2199686935424805 | Acc: 0.1837000000000000.
Final evaluation: Validation loss: 2.2199686935424805
Final evaluation: Validation metric: 0.18369999527931213

(2.2199686935424805, nan, tensor(0.1837))

14.10.4 Evaluation of the Tuned Architecture

The following code trains the model `model_spot`.

If `path` is set to a filename, e.g., `path = "model_spot_trained.pt"`, the weights of the trained model will be saved to this file.

If `path` is set to a filename, e.g., `path = "model_spot_trained.pt"`, the weights of the trained model will be loaded from this file.

```
train_tuned(net=model_spot, train_dataset=train,
            loss_function=fun_control["loss_function"],
            metric=fun_control["metric_torch"],
            shuffle=True,
            device = fun_control["device"],
            path=None,
            task=fun_control["task"],)
test_tuned(net=model_spot, test_dataset=test,
            shuffle=False,
            loss_function=fun_control["loss_function"],
            metric=fun_control["metric_torch"],
            device = fun_control["device"],
            task=fun_control["task"],)
```

Epoch: 1 |

MulticlassAccuracy: 0.4460499882698059 | Loss: 1.5166250946044921 | Acc: 0.4460500000000000.
Epoch: 2 |

```

MulticlassAccuracy: 0.4973500072956085 | Loss: 1.3976579611778259 | Acc: 0.4973500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.5412999987602234 | Loss: 1.2810937417030335 | Acc: 0.5413000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5569000244140625 | Loss: 1.2339107001304626 | Acc: 0.5569000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5667499899864197 | Loss: 1.2280268907546996 | Acc: 0.5667500000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5781000256538391 | Loss: 1.1878850612640381 | Acc: 0.5780999999999999.
Epoch: 7 |

MulticlassAccuracy: 0.5874999761581421 | Loss: 1.1821841226577758 | Acc: 0.5875000000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5930500030517578 | Loss: 1.1730224997520446 | Acc: 0.5930500000000000.
Epoch: 9 |

MulticlassAccuracy: 0.5921000242233276 | Loss: 1.1829405002593993 | Acc: 0.5921000000000000.
Epoch: 10 |

MulticlassAccuracy: 0.5839499831199646 | Loss: 1.2501484612464904 | Acc: 0.5839500000000000.
Epoch: 11 |

MulticlassAccuracy: 0.5913500189781189 | Loss: 1.2072612472534179 | Acc: 0.5913500000000000.
Early stopping at epoch 10
Returned to Spot: Validation loss: 1.2072612472534179

MulticlassAccuracy: 0.5849000215530396 | Loss: 1.2212051355038969 | Acc: 0.5849000000000000.
Final evaluation: Validation loss: 1.2212051355038969
Final evaluation: Validation metric: 0.5849000215530396
-----

(1.2212051355038969, nan, tensor(0.5849))

```

14.10.5 Detailed Hyperparameter Plots

The contour plots in this section visualize the interactions of the three most important hyperparameters. Since some of these hyperparameters take factorial or integer values, sometimes step-like fitness landscapes (or response surfaces) are generated. SPOT draws the interactions of the main hyperparameters by default. It is also possible to visualize all interactions.

```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

```
l1: 69.41527482095373
batch_size: 100.0
optimizer: 37.528552455791356
```

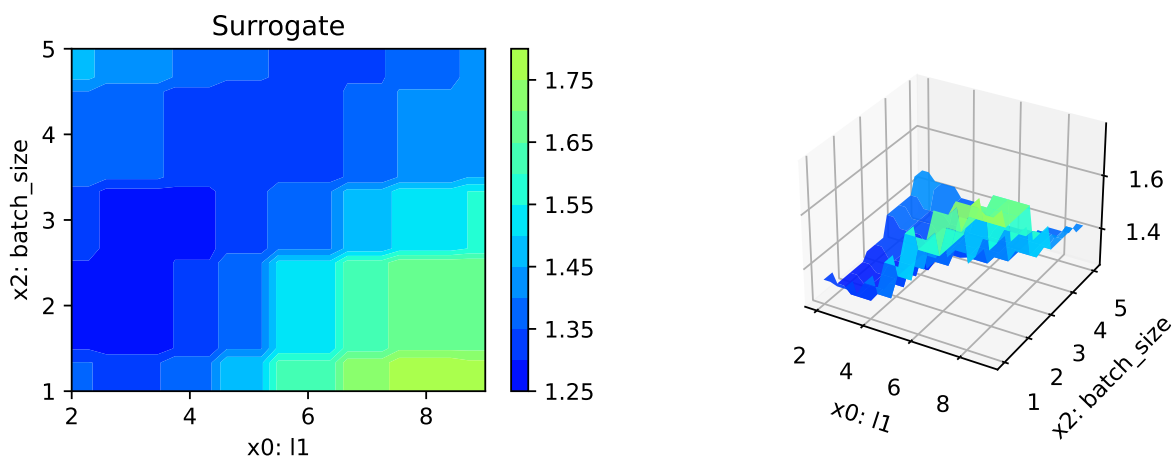
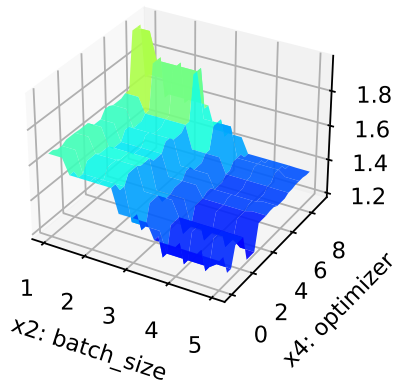
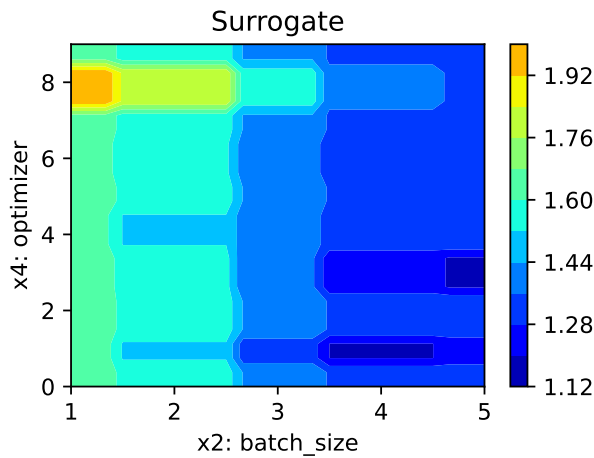
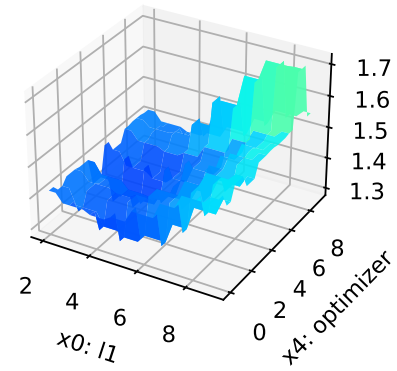
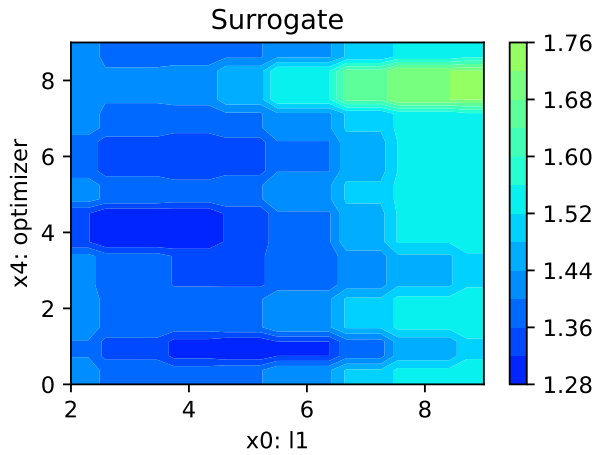


Figure 14.5: Contour plots.



The figures (`?@fig-contour`) show the contour plots of the loss as a function of the hyperparameters. These plots are very helpful for benchmark studies and for understanding neural networks. `spotPython` provides additional tools for a visual inspection of the results and give valuable insights into the hyperparameter tuning process. This is especially useful for model explainability, transparency, and trustworthiness. In addition to the contour plots, `?@fig-parallel` shows the parallel plot of the hyperparameters.

```
spot_tuner.parallel_plot()
```

Unable to display output for mime type(s): text/html

Parallel coordinates plots

Unable to display output for mime type(s): text/html

14.11 Summary and Outlook

This tutorial presents the hyperparameter tuning open source software `spotPython` for `PyTorch`. To show its basic features, a comparison with the “official” `PyTorch` hyperparameter tuning tutorial (PyTorch 2023a) is presented. Some of the advantages of `spotPython` are:

- Numerical and categorical hyperparameters.
- Powerful surrogate models.
- Flexible approach and easy to use.
- Simple JSON files for the specification of the hyperparameters.
- Extension of default and user specified network classes.
- Noise handling techniques.
- Interaction with `tensorboard`.

Currently, only rudimentary parallel and distributed neural network training is possible, but these capabilities will be extended in the future. The next version of `spotPython` will also include a more detailed documentation and more examples.

! Important

Important: This tutorial does not present a complete benchmarking study (Bartz-Beielstein et al. 2020). The results are only preliminary and highly dependent on the local configuration (hard- and software). Our goal is to provide a first impression of the performance of the hyperparameter tuning package `spotPython`. To demonstrate its capabilities, a quick comparison with `ray[tune]` was performed. `ray[tune]` was chosen, because it is presented as “an industry standard tool for distributed hyperparameter tuning.” The results should be interpreted with care.

14.12 Appendix

14.12.1 Sample Output From Ray Tune’s Run

The output from `ray[tune]` could look like this (PyTorch 2023b):

```
Number of trials: 10 (10 TERMINATED)
-----+-----+-----+-----+-----+-----+-----+
|  11 |  12 |           lr | batch_size |   loss | accuracy | training_iteration |
```


15 HPT: sklearn RandomForestClassifier VBDP Data

This document refers to the following software versions:

- python: 3.10.10

```
pip list | grep "spot[RiverPython]"
```

spotPython	0.2.52
spotRiver	0.0.94

Note: you may need to restart the kernel to use updated packages.

spotPython can be installed via pip. Alternatively, the source code can be downloaded from gitHub: <https://github.com/sequential-parameter-optimization/spotPython>.

```
!pip install spotPython
```

- Uncomment the following lines if you want to for (re-)installation the latest version of spotPython from gitHub.

```
# import sys
# !{sys.executable} -m pip install --upgrade build
# !{sys.executable} -m pip install --upgrade --force-reinstall spotPython
```

15.1 Step 1: Setup

Before we consider the detailed experimental setup, we select the parameters that affect run time and the initial design size.

```

MAX_TIME = 1
INIT_SIZE = 5
ORIGINAL = False

import os
import copy
import socket
from datetime import datetime
from dateutil.tz import tzlocal
start_time = datetime.now(tzlocal())
HOSTNAME = socket.gethostname().split(".")[0]
experiment_name = '16-rf-sklearn' + "_" + HOSTNAME + "_" + str(MAX_TIME) + "min_" + str(INIT_SIZE)
experiment_name = experiment_name.replace(':', '-')
print(experiment_name)
if not os.path.exists('./figures'):
    os.makedirs('./figures')

```

16-rf-sklearn_maans03_1min_5init_2023-07-03_12-53-52

```

import warnings
warnings.filterwarnings("ignore")

```

15.2 Step 2: Initialization of the Empty fun_control Dictionary

 Caution: Tensorboard does not work under Windows

- Since tensorboard does not work under Windows, we recommend setting the parameter `tensorboard_path` to `None` if you are working under Windows.

```

from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="classification",
    tensorboard_path="runs/16_spot_hpt_sklearn_classification")

```

15.3 Step 3: PyTorch Data Loading

15.3.1 Load Data: Classification VBDP

```
import pandas as pd
if ORIGINAL == True:
    train_df = pd.read_csv('./data/VBDP/trainnn.csv')
    test_df = pd.read_csv('./data/VBDP/testtt.csv')
else:
    train_df = pd.read_csv('./data/VBDP/train.csv')
    # remove the id column
    train_df = train_df.drop(columns=['id'])

from sklearn.preprocessing import OrdinalEncoder
n_samples = train_df.shape[0]
n_features = train_df.shape[1] - 1
target_column = "prognosis"
# Encoder our prognosis labels as integers for easier decoding later
enc = OrdinalEncoder()
train_df[target_column] = enc.fit_transform(train_df[[target_column]])
train_df.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
print(train_df.shape)
train_df.head()
```

(707, 65)

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19
0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0
1	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0
2	0.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0
3	0.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

The full data set `train_df` 64 features. The target column is labeled as `prognosis`.

15.3.2 Holdout Train and Test Data

We split out a hold-out test set (25% of the data) so we can calculate an example MAP@K

```

import numpy as np
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(train_df.drop(target_column, axis=1),
                                                    random_state=42,
                                                    test_size=0.25,
                                                    stratify=train_df[target_column])

train = pd.DataFrame(np.hstack((X_train, np.array(y_train).reshape(-1, 1))))
test = pd.DataFrame(np.hstack((X_test, np.array(y_test).reshape(-1, 1))))
train.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
test.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
print(train.shape)
print(test.shape)
train.head()

```

(530, 65)

(177, 65)

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0
2	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	1.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0
4	0.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0

```

# add the dataset to the fun_control
fun_control.update({"data": train_df, # full dataset,
                  "train": train,
                  "test": test,
                  "n_samples": n_samples,
                  "target_column": target_column})

```

15.4 Step 4: Specification of the Preprocessing Model

Data preprocesssing can be very simple, e.g., you can ignore it. Then you would choose the `prep_model` “None”:

```

prep_model = None
fun_control.update({"prep_model": prep_model})

```

A default approach for numerical data is the `StandardScaler` (mean 0, variance 1). This can be selected as follows:

```
# prep_model = StandardScaler()
# fun_control.update({"prep_model": prep_model})
```

Even more complicated pre-processing steps are possible, e.g., the following pipeline:

```
# categorical_columns = []
# one_hot_encoder = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
# prep_model = ColumnTransformer(
#     transformers=[
#         ("categorical", one_hot_encoder, categorical_columns),
#     ],
#     remainder=StandardScaler(),
# )
```

15.5 Step 5: Select Model (algorithm) and `core_model_hyper_dict`

The selection of the algorithm (ML model) that should be tuned is done by specifying the its name from the `sklearn` implementation. For example, the SVC support vector machine classifier is selected as follows:

```
fun_control = add_core_model_to_fun_control(SVC, fun_control, SklearnHyperDict)
```

Other `core_models` are, e.g.,:

- `RidgeCV`
- `GradientBoostingRegressor`
- `ElasticNet`
- `RandomForestClassifier`
- `LogisticRegression`
- `KNeighborsClassifier`
- `RandomForestClassifier`
- `GradientBoostingClassifier`
- `HistGradientBoostingClassifier`

We will use the `RandomForestClassifier` classifier in this example.

```

from sklearn.linear_model import RidgeCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import ElasticNet
from spotPython.hyperparameters.values import add_core_model_to_fun_control
from spotPython.data.sklearn_hyper_dict import SklearnHyperDict
from spotPython.fun.hypersklearn import HyperSklearn

```

```

# core_model = RidgeCV
# core_model = GradientBoostingRegressor
# core_model = ElasticNet
core_model = RandomForestClassifier
# core_model = SVC
# core_model = LogisticRegression
# core_model = KNeighborsClassifier
# core_model = GradientBoostingClassifier
fun_control = add_core_model_to_fun_control(core_model=core_model,
                                           fun_control=fun_control,
                                           hyper_dict=SklearnHyperDict,
                                           filename=None)

```

Now `fun_control` has the information from the JSON file. The available hyperparameters are:

```

print(*fun_control["core_model_hyper_dict"].keys(), sep="\n")

```

```

n_estimators
criterion
max_depth
min_samples_split
min_samples_leaf
min_weight_fraction_leaf
max_features
max_leaf_nodes
min_impurity_decrease
bootstrap
oob_score

```

15.6 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model`

15.6.1 Modify hyperparameter of type numeric and integer (boolean)

Numeric and boolean values can be modified using the `modify_hyper_parameter_bounds` method. For example, to change the `tol` hyperparameter of the SVC model to the interval `[1e-3, 1e-2]`, the following code can be used:

```
fun_control = modify_hyper_parameter_bounds(fun_control, "tol", bounds=[1e-3, 1e-2])
```

```
from spotPython.hyperparameters.values import modify_hyper_parameter_bounds
# fun_control = modify_hyper_parameter_bounds(fun_control, "tol", bounds=[1e-3, 1e-2])
```

15.6.2 Modify hyperparameter of type factor

`spotPython` provides functions for modifying the hyperparameters, their bounds and factors as well as for activating and de-activating hyperparameters without re-compilation of the Python source code. These functions were described in Section 14.6.

Factors can be modified with the `modify_hyper_parameter_levels` function. For example, to exclude the `sigmoid` kernel from the tuning, the `kernel` hyperparameter of the SVC model can be modified as follows:

```
fun_control = modify_hyper_parameter_levels(fun_control, "kernel", ["linear", "rbf"])
```

The new setting can be controlled via:

```
fun_control["core_model_hyper_dict"]["kernel"]
```

```
from spotPython.hyperparameters.values import modify_hyper_parameter_levels
# XGBoost:
# fun_control = modify_hyper_parameter_levels(fun_control, "loss", ["log_loss"])
```

i Note: RandomForestClassifier and Out-of-bag Estimation

Since `oob_score` requires the `bootstrap` hyperparameter to `True`, we set the `oob_score` parameter to `False`. The `oob_score` is later discussed in Section 15.7.3.


```
fun_control = modify_hyper_parameter_bounds(fun_control, "bootstrap", bounds=[0, 1])
fun_control = modify_hyper_parameter_bounds(fun_control, "oob_score", bounds=[0, 0])
```

15.6.3 Optimizers

Optimizers are described in Section [14.6.1](#).

15.6.4 Selection of the Objective: Metric and Loss Functions

- Machine learning models are optimized with respect to a metric, for example, the accuracy function.
- Deep learning, e.g., neural networks are optimized with respect to a loss function, for example, the `cross_entropy` function and evaluated with respect to a metric, for example, the accuracy function.

15.7 Step 7: Selection of the Objective (Loss) Function

The loss function, that is usually used in deep learning for optimizing the weights of the net, is stored in the `fun_control` dictionary as `"loss_function"`.

15.7.1 Metric Function

There are two different types of metrics in `spotPython`:

1. `"metric_river"` is used for the river based evaluation via `eval_oml_iter_progressive`.
2. `"metric_sklearn"` is used for the sklearn based evaluation.

We will consider multi-class classification metrics, e.g., `mapk_score` and `top_k_accuracy_score`.

Predict Probabilities

In this multi-class classification example the machine learning algorithm should return the probabilities of the specific classes (`"predict_proba"`) instead of the predicted values.

We set `"predict_proba"` to `True` in the `fun_control` dictionary.

15.7.1.1 The MAPK Metric

To select the MAPK metric, the following two entries can be added to the `fun_control` dictionary:

```
"metric_sklearn": mapk_score"  
"metric_params": {"k": 3}.
```

15.7.1.2 Other Metrics

Alternatively, other metrics for multi-class classification can be used, e.g., `* top_k_accuracy_score` or `* roc_auc_score`

The metric `roc_auc_score` requires the parameter `"multi_class"`, e.g.,

```
"multi_class": "ovr".
```

This is set in the `fun_control` dictionary.

Weights

`spotPython` performs a minimization, therefore, metrics that should be maximized have to be multiplied by -1. This is done by setting `"weights"` to -1.

- The complete setup for the metric in our example is:

```
from spotPython.utils.metrics import mapk_score  
fun_control.update({  
    "weights": -1,  
    "metric_sklearn": mapk_score,  
    "predict_proba": True,  
    "metric_params": {"k": 3},  
})
```

15.7.2 Evaluation on Hold-out Data

- The default method for computing the performance is `"eval_holdout"`.
- Alternatively, cross-validation can be used for every machine learning model.
- Specifically for RandomForests, the OOB-score can be used.

```
fun_control.update({
    "eval": "train_hold_out",
})
```

15.7.3 OOB Score

Using the OOB-Score is a very efficient way to estimate the performance of a random forest classifier. The OOB-Score is calculated on the training data and does not require a hold-out test set. If the OOB-Score is used, the key “eval” in the `fun_control` dictionary should be set to `"oob_score"` as shown below.

i OOB-Score

In addition to setting the key `"eval"` in the `fun_control` dictionary to `"oob_score"`, the keys `"oob_score"` and `"bootstrap"` have to be set to `True`, because the OOB-Score requires the bootstrap method.

- Uncomment the following lines to use the OOB-Score:

```
fun_control.update({
    "eval": "eval_oob_score",
})
fun_control = modify_hyper_parameter_bounds(fun_control, "bootstrap", bounds=[1, 1])
fun_control = modify_hyper_parameter_bounds(fun_control, "oob_score", bounds=[1, 1])
```

15.7.3.1 Cross Validation

Instead of using the OOB-score, the classical cross validation can be used. The number of folds is set by the key `"k_folds"`. For example, to use 5-fold cross validation, the key `"k_folds"` is set to 5. Uncomment the following line to use cross validation:

```
# fun_control.update({
#     "eval": "train_cv",
#     "k_folds": 10,
# })
```

15.8 Step 8: Calling the SPOT Function

15.8.1 Preparing the SPOT Call

- Get types and variable names as well as lower and upper bounds for the hyperparameters.

```
# extract the variable types, names, and bounds
from spotPython.hyperparameters.values import (get_bound_values,
        get_var_name,
        get_var_type,)
var_type = get_var_type(fun_control)
var_name = get_var_name(fun_control)
fun_control.update({"var_type": var_type,
                   "var_name": var_name})
lower = get_bound_values(fun_control, "lower")
upper = get_bound_values(fun_control, "upper")

from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control))
```

name	type	default	lower	upper	transform
n_estimators	int	7	5	10	transform_power_2_int
criterion	factor	gini	0	2	None
max_depth	int	10	1	20	transform_power_2_int
min_samples_split	int	2	2	100	None
min_samples_leaf	int	1	1	25	None
min_weight_fraction_leaf	float	0.0	0	0.01	None
max_features	factor	sqrt	0	1	transform_none_to_None
max_leaf_nodes	int	10	7	12	transform_power_2_int
min_impurity_decrease	float	0.0	0	0.01	None
bootstrap	factor	1	1	1	None
oob_score	factor	0	1	1	None

15.8.2 The Objective Function

The objective function is selected next. It implements an interface from `sklearn`'s training, validation, and testing methods to `spotPython`.

```
from spotPython.fun.hypersklearn import HyperSklearn
fun = HyperSklearn().fun_sklearn
```

15.8.3 Run the Spot Optimizer

- Run SPOT for approx. x mins (`max_time`).
- Note: the run takes longer, because the evaluation time of initial design (here: `init_size`, 20 points) is not considered.

```
from spotPython.hyperparameters.values import get_default_hyperparameters_as_array
hyper_dict=SklearnHyperDict().load()
X_start = get_default_hyperparameters_as_array(fun_control, hyper_dict)
X_start
```

```
array([[ 7.,  0., 10.,  2.,  1.,  0.,  0., 10.,  0.,  1.,  0.]])
```

```
import numpy as np
from spotPython.spot import spot
from math import inf
spot_tuner = spot.Spot(fun=fun,
                      lower = lower,
                      upper = upper,
                      fun_evals = inf,
                      fun_repeats = 1,
                      max_time = MAX_TIME,
                      noise = False,
                      tolerance_x = np.sqrt(np.spacing(1)),
                      var_type = var_type,
                      var_name = var_name,
                      infill_criterion = "y",
                      n_points = 1,
                      seed=123,
                      log_level = 50,
                      show_models= False,
                      show_progress= True,
                      fun_control = fun_control,
                      design_control={"init_size": INIT_SIZE,
                                    "repeats": 1},
                      surrogate_control={"noise": True,
                                       "cod_type": "norm",
```

```

        "min_theta": -4,
        "max_theta": 3,
        "n_theta": len(var_name),
        "model_fun_evals": 10_000,
        "log_level": 50
    })

spot_tuner.run(X_start=X_start)

```

```

spotPython tuning: -0.34276729559748426 [-----] 3.65%

spotPython tuning: -0.34276729559748426 [-----] 4.88%

spotPython tuning: -0.34276729559748426 [#-----] 6.12%

spotPython tuning: -0.35660377358490564 [#-----] 7.50%

spotPython tuning: -0.35660377358490564 [#-----] 9.03%

spotPython tuning: -0.35660377358490564 [#-----] 10.22%

spotPython tuning: -0.35660377358490564 [#-----] 12.81%

spotPython tuning: -0.35660377358490564 [##-----] 16.14%

spotPython tuning: -0.35660377358490564 [##-----] 19.29%

spotPython tuning: -0.35660377358490564 [##-----] 21.95%

spotPython tuning: -0.35660377358490564 [##-----] 24.80%

spotPython tuning: -0.35660377358490564 [###-----] 27.44%

spotPython tuning: -0.35660377358490564 [###-----] 31.70%

spotPython tuning: -0.35660377358490564 [####-----] 37.21%

spotPython tuning: -0.35660377358490564 [####-----] 43.71%

```

```
spotPython tuning: -0.35660377358490564 [#####-----] 50.79%

spotPython tuning: -0.35660377358490564 [#####-----] 56.39%

spotPython tuning: -0.35660377358490564 [#####-----] 62.30%

spotPython tuning: -0.35660377358490564 [#####-----] 68.20%

spotPython tuning: -0.35660377358490564 [#####-----] 75.81%

spotPython tuning: -0.35660377358490564 [#####-----] 80.00%

spotPython tuning: -0.35660377358490564 [#####-----] 86.34%

spotPython tuning: -0.35660377358490564 [#####-----] 91.83%

spotPython tuning: -0.35660377358490564 [#####-----] 95.98%

spotPython tuning: -0.35660377358490564 [#####-----] 100.00% Done...

<spotPython.spot.spot.Spot at 0x18c6df940>
```

15.9 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section [14.9](#), see also the description in the documentation: [Tensorboard](#).

15.10 Step 10: Results

After the hyperparameter tuning run is finished, the progress of the hyperparameter tuning can be visualized. The following code generates the progress plot from `?@fig-progress`.

```
spot_tuner.plot_progress(log_y=False,
                        filename="./figures/" + experiment_name+"_progress.png")
```

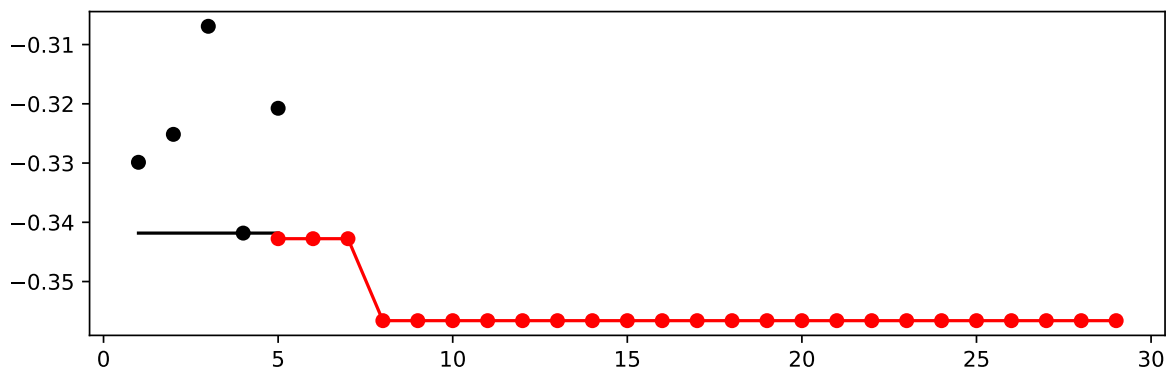


Figure 15.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- Print the results

```
print(gen_design_table(fun_control=fun_control,
                      spot=spot_tuner))
```

name	type	default	lower	upper	tuned
n_estimators	int	7	5.0	10.0	7.0
criterion	factor	gini	0.0	2.0	1.0
max_depth	int	10	1.0	20.0	15.0
min_samples_split	int	2	2.0	100.0	10.0
min_samples_leaf	int	1	1.0	25.0	5.0
min_weight_fraction_leaf	float	0.0	0.0	0.01	0.009830186235311258
max_features	factor	sqrt	0.0	1.0	0.0
max_leaf_nodes	int	10	7.0	12.0	10.0
min_impurity_decrease	float	0.0	0.0	0.01	0.009438663023216103
bootstrap	factor	1	1.0	1.0	1.0
oob_score	factor	0	1.0	1.0	1.0

15.10.1 Show variable importance

```
spot_tuner.plot_importance(threshold=0.025, filename="./figures/" + experiment_name+"_impo
```

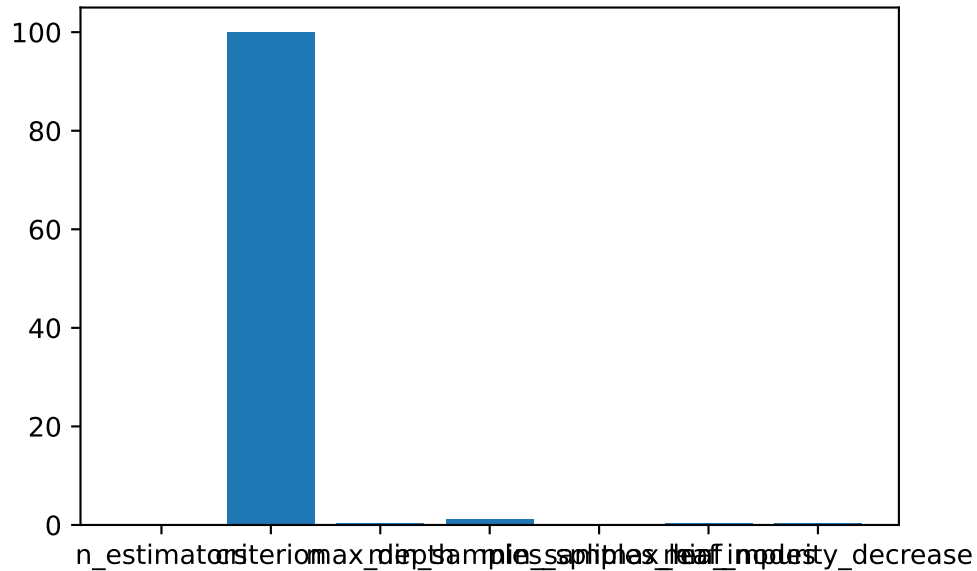


Figure 15.2: Variable importance plot, threshold 0.025.

15.10.2 Get Default Hyperparameters

```
from spotPython.hyperparameters.values import get_default_values, transform_hyper_parameter_values_default = get_default_values(fun_control) values_default = transform_hyper_parameter_values(fun_control=fun_control, hyper_parameter_values_default
```

```
{'n_estimators': 128,
 'criterion': 'gini',
 'max_depth': 1024,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'min_weight_fraction_leaf': 0.0,
 'max_features': 'sqrt',
 'max_leaf_nodes': 1024,
 'min_impurity_decrease': 0.0,
```

```
'bootstrap': 1,
'oob_score': 0}
```

```
from sklearn.pipeline import make_pipeline
model_default = make_pipeline(fun_control["prep_model"], fun_control["core_model"](**value
model_default
```

```
Pipeline(steps=[('nonetype', None),
                  ('randomforestclassifier',
                   RandomForestClassifier(bootstrap=1, max_depth=1024,
                                         max_leaf_nodes=1024, n_estimators=128,
                                         oob_score=0))])
```

15.10.3 Get SPOT Results

```
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
print(X)
```

```
[[7.00000000e+00 1.00000000e+00 1.50000000e+01 1.00000000e+01
 5.00000000e+00 9.83018624e-03 0.00000000e+00 1.00000000e+01
 9.43866302e-03 1.00000000e+00 1.00000000e+00]]
```

```
from spotPython.hyperparameters.values import assign_values, return_conf_list_from_var_dict
v_dict = assign_values(X, fun_control["var_name"])
return_conf_list_from_var_dict(var_dict=v_dict, fun_control=fun_control)
```

```
[{'n_estimators': 128,
  'criterion': 'entropy',
  'max_depth': 32768,
  'min_samples_split': 10,
  'min_samples_leaf': 5,
  'min_weight_fraction_leaf': 0.009830186235311258,
  'max_features': 'sqrt',
  'max_leaf_nodes': 1024,
  'min_impurity_decrease': 0.009438663023216103,
  'bootstrap': 1,
  'oob_score': 1}]
```

```

from spotPython.hyperparameters.values import get_one_sklearn_model_from_X
model_spot = get_one_sklearn_model_from_X(X, fun_control)
model_spot

```

```

RandomForestClassifier(bootstrap=1, criterion='entropy', max_depth=32768,
                        max_leaf_nodes=1024,
                        min_impurity_decrease=0.009438663023216103,
                        min_samples_leaf=5, min_samples_split=10,
                        min_weight_fraction_leaf=0.009830186235311258,
                        n_estimators=128, oob_score=1)

```

15.10.4 Evaluate SPOT Results

- Fetch the data.

```

from spotPython.utils.convert import get_Xy_from_df
X_train, y_train = get_Xy_from_df(fun_control["train"], fun_control["target_column"])
X_test, y_test = get_Xy_from_df(fun_control["test"], fun_control["target_column"])
X_test.shape, y_test.shape

```

```
((177, 64), (177,))
```

- Fit the model with the tuned hyperparameters. This gives one result:

```

model_spot.fit(X_train, y_train)
y_pred = model_spot.predict_proba(X_test)
res = mapk_score(y_true=y_test, y_pred=y_pred, k=3)
res

```

```
0.35216572504708094
```

```

def repeated_eval(n, model):
    res_values = []
    for i in range(n):
        model.fit(X_train, y_train)
        y_pred = model.predict_proba(X_test)
        res = mapk_score(y_true=y_test, y_pred=y_pred, k=3)
        res_values.append(res)
    mean_res = np.mean(res_values)

```

```

print(f"mean_res: {mean_res}")
std_res = np.std(res_values)
print(f"std_res: {std_res}")
min_res = np.min(res_values)
print(f"min_res: {min_res}")
max_res = np.max(res_values)
print(f"max_res: {max_res}")
median_res = np.median(res_values)
print(f"median_res: {median_res}")
return mean_res, std_res, min_res, max_res, median_res

```

15.10.5 Handling Non-deterministic Results

- Because the model is non-deterministic, we perform $n = 30$ runs and calculate the mean and standard deviation of the performance metric.

```
_ = repeated_eval(30, model_spot)
```

```

mean_res: 0.35699937225360956
std_res: 0.009657098348021476
min_res: 0.3380414312617702
max_res: 0.37758945386064036
median_res: 0.35687382297551784

```

15.10.6 Evaluation of the Default Hyperparameters

```
model_default.fit(X_train, y_train)["randomforestclassifier"]
```

```

RandomForestClassifier(bootstrap=1, max_depth=1024, max_leaf_nodes=1024,
                        n_estimators=128, oob_score=0)

```

- One evaluation of the default hyperparameters is performed on the hold-out test set.

```

y_pred = model_default.predict_proba(X_test)
mapk_score(y_true=y_test, y_pred=y_pred, k=3)

```

```
0.3418079096045198
```

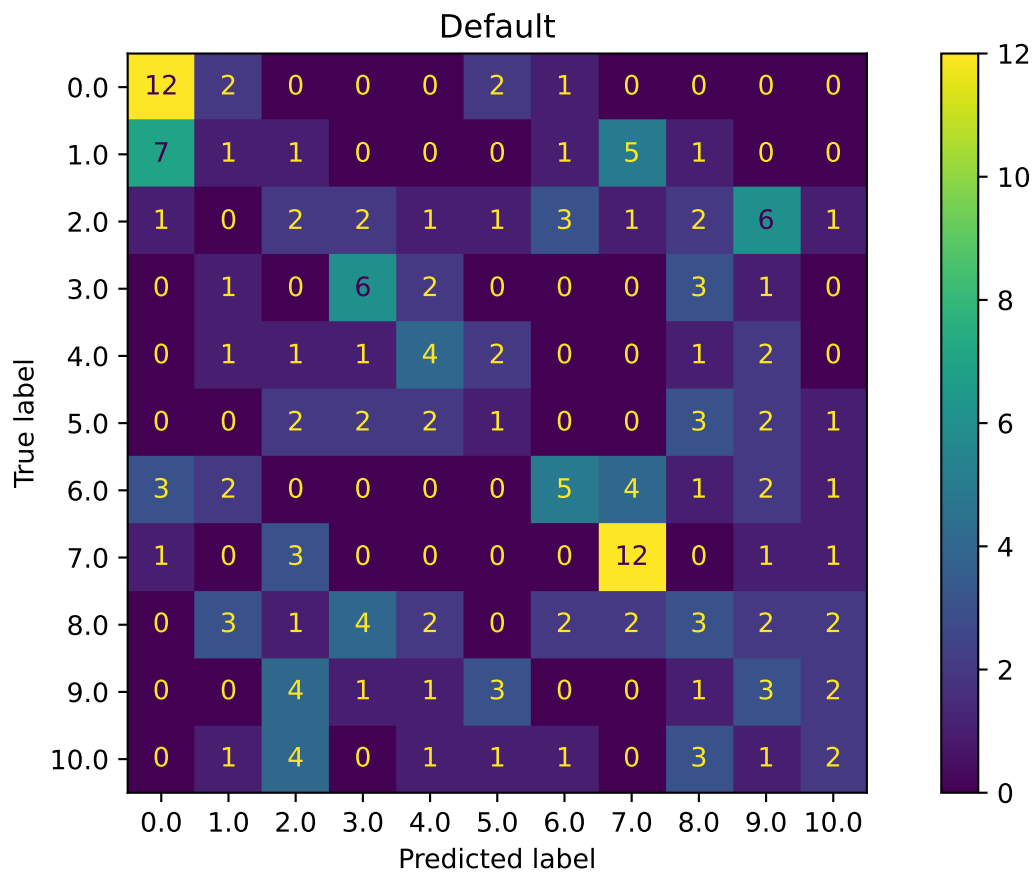
Since one single evaluation is not meaningful, we perform, similar to the evaluation of the SPOT results, $n = 30$ runs of the default setting and calculate the mean and standard deviation of the performance metric.

```
_ = repeated_eval(30, model_default)
```

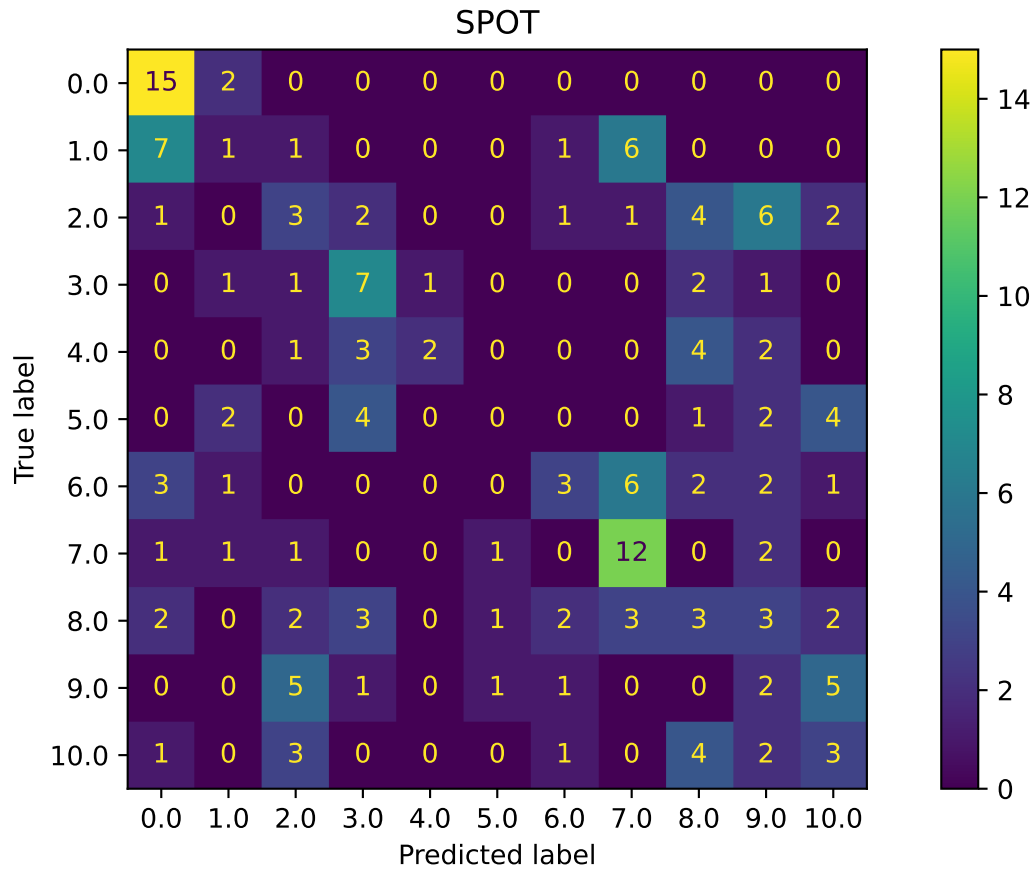
```
mean_res: 0.34328311362209674
std_res: 0.014647524137942517
min_res: 0.3135593220338983
max_res: 0.37382297551789073
median_res: 0.3418079096045198
```

15.10.7 Plot: Compare Predictions

```
from spotPython.plot.validation import plot_confusion_matrix
plot_confusion_matrix(model_default, fun_control, title = "Default")
```



```
plot_confusion_matrix(model_spot, fun_control, title="SPOT")
```



```
min(spot_tuner.y), max(spot_tuner.y)
```

```
(-0.35660377358490564, -0.27861635220125786)
```

15.10.8 Cross-validated Evaluations

```
from spotPython.sklearn.traintest import evaluate_cv
fun_control.update({
    "eval": "train_cv",
    "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)
```

```
(0.361006289308176, None)
```

```

fun_control.update({
    "eval": "test_cv",
    "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)

```

(0.32293028322440087, None)

- This is the evaluation that will be used in the comparison:

```

fun_control.update({
    "eval": "data_cv",
    "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)

```

(0.35895036887994636, None)

15.10.9 Detailed Hyperparameter Plots

```

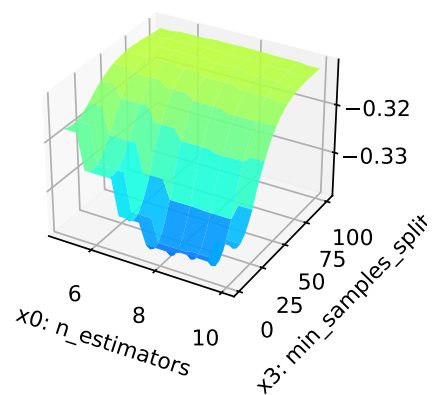
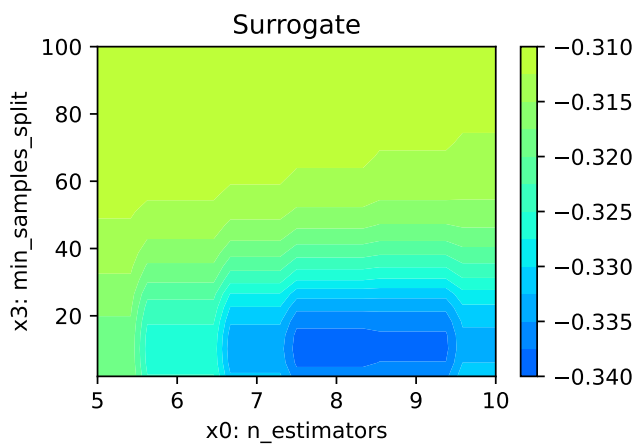
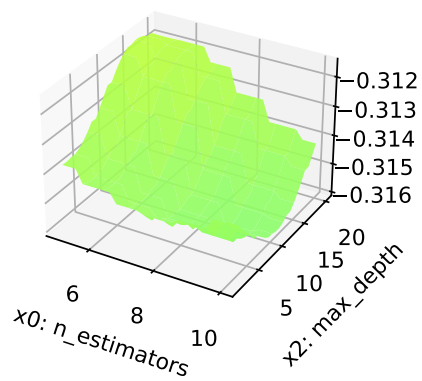
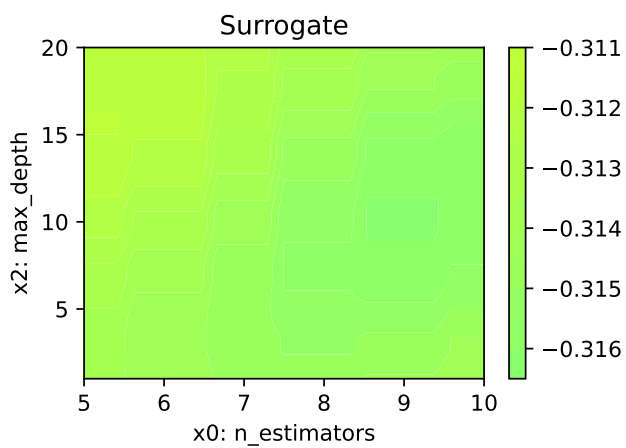
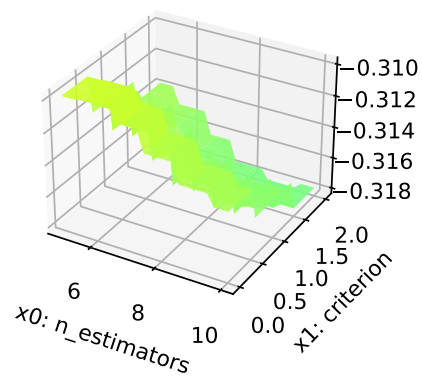
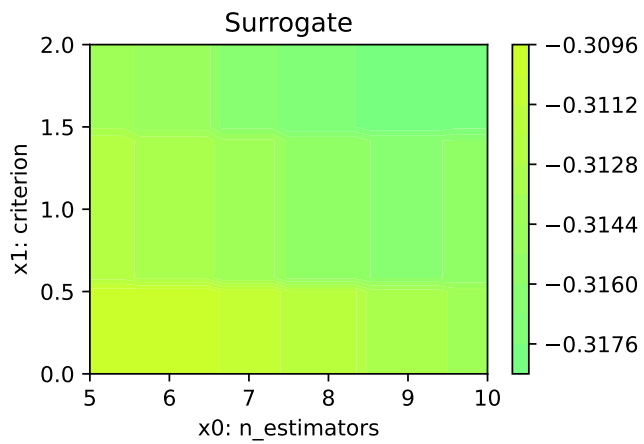
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)

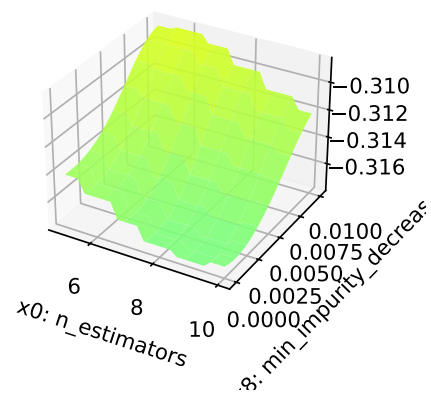
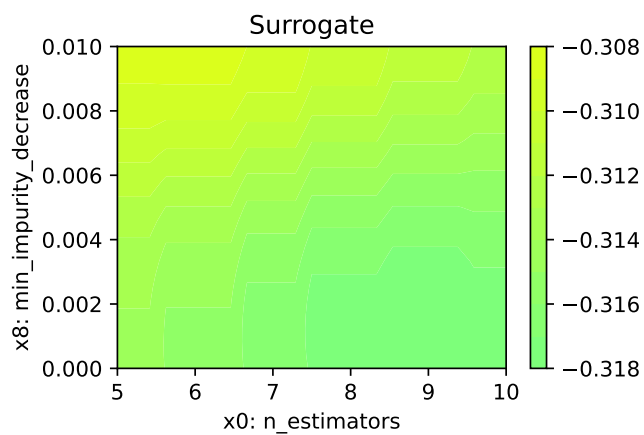
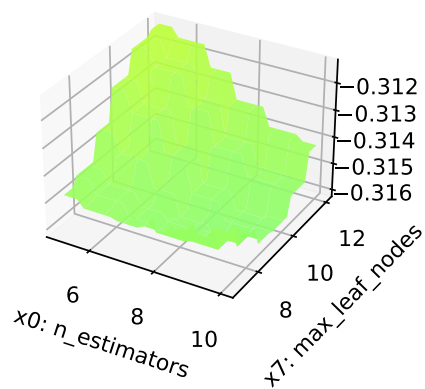
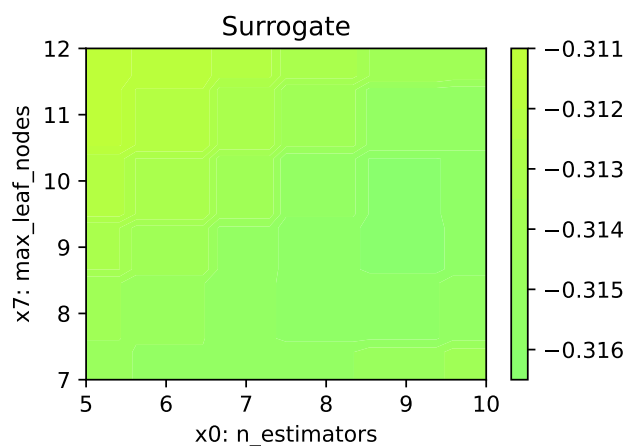
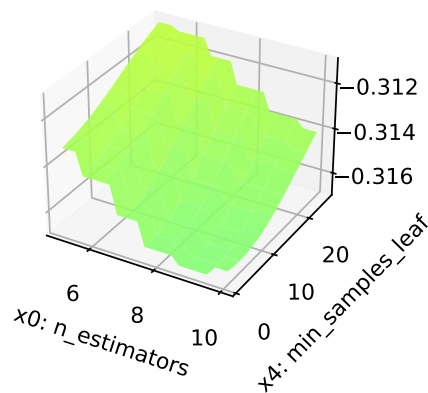
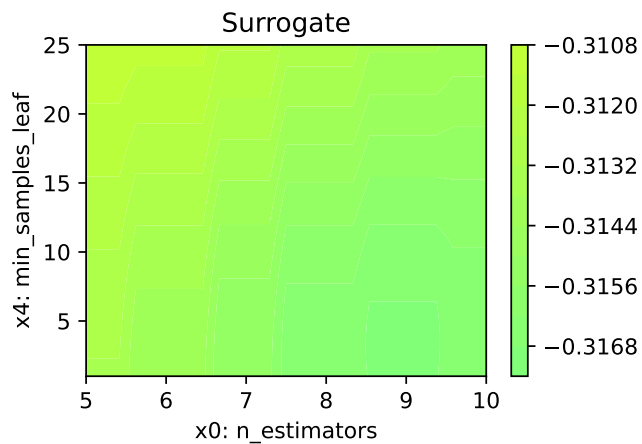
```

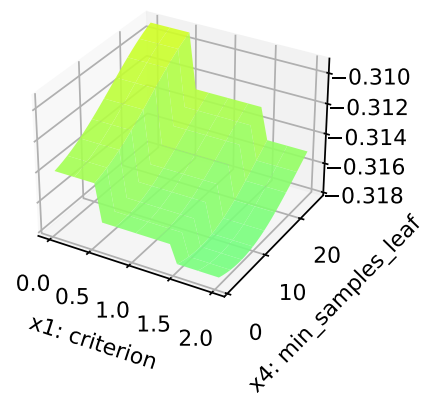
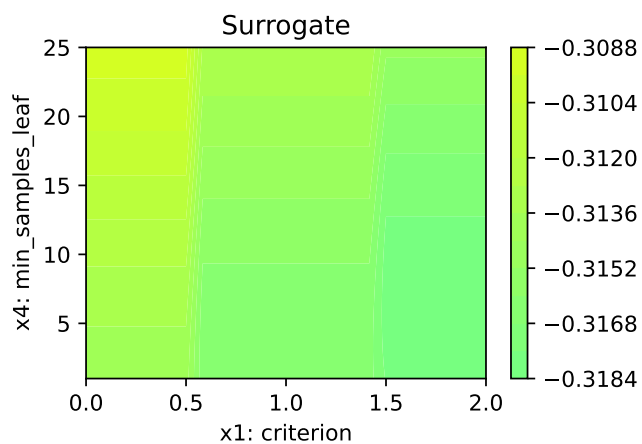
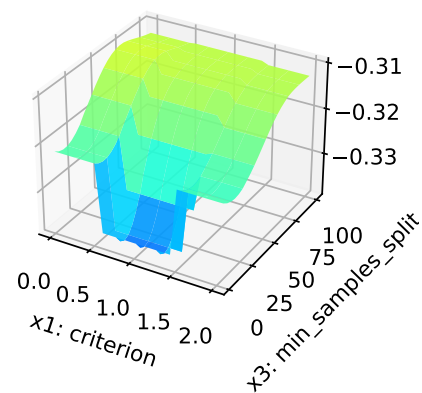
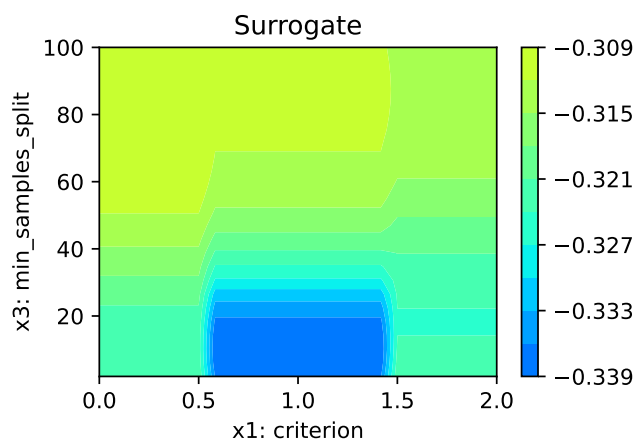
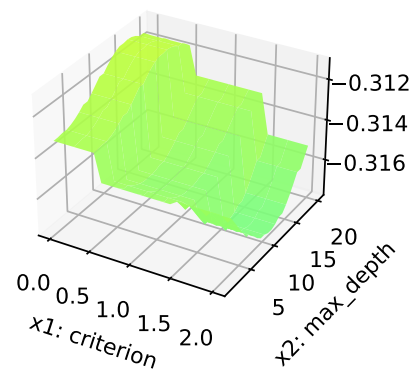
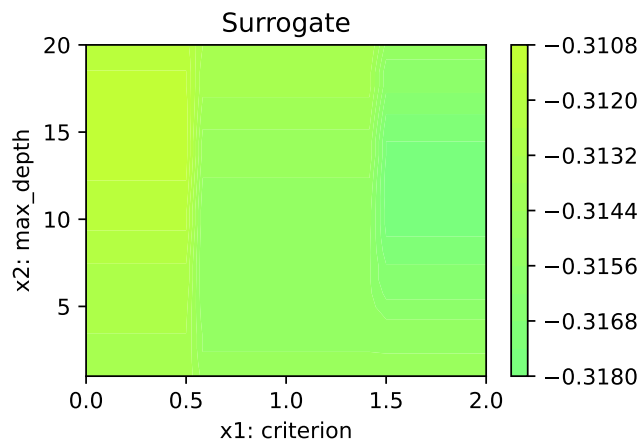
```

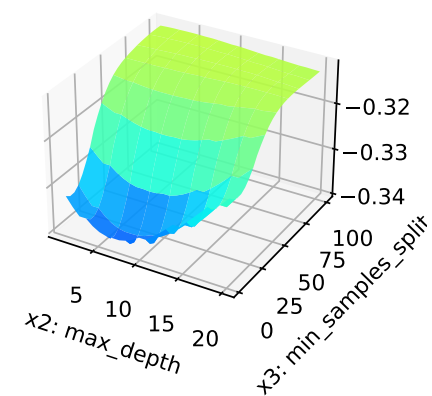
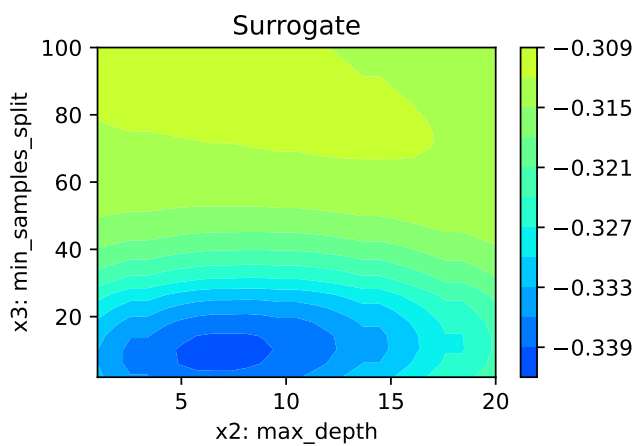
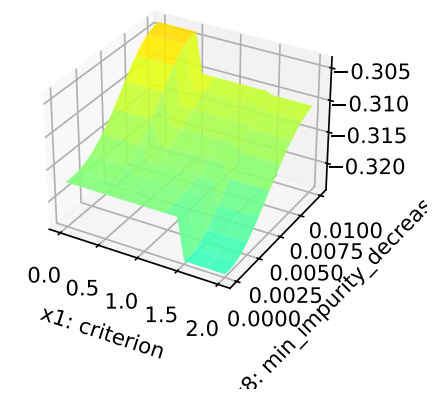
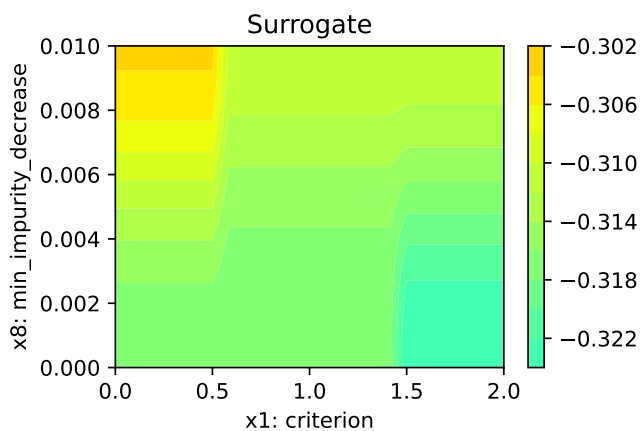
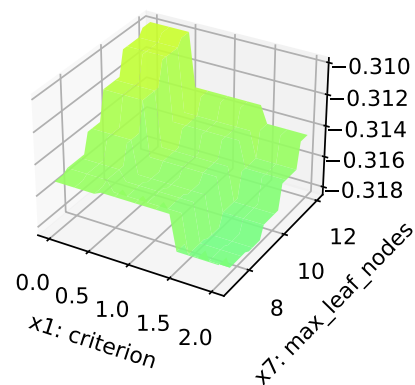
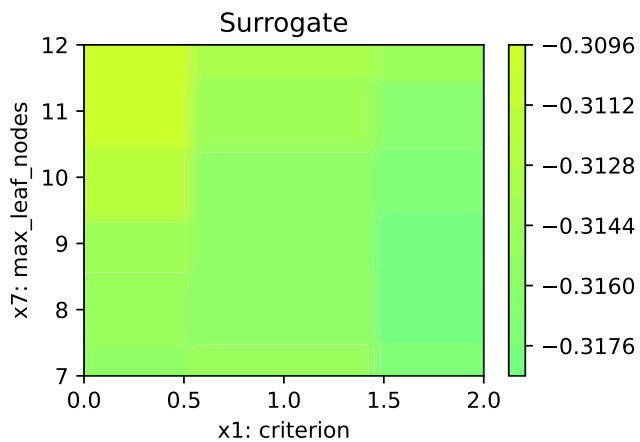
n_estimators: 0.13783730480984077
criterion: 100.0
max_depth: 0.3689792841238967
min_samples_split: 1.2017863048350383
min_samples_leaf: 0.1079389105014468
max_leaf_nodes: 0.32630846220040316
min_impurity_decrease: 0.3663329270575742

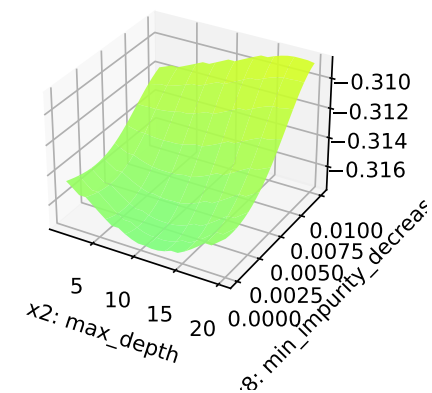
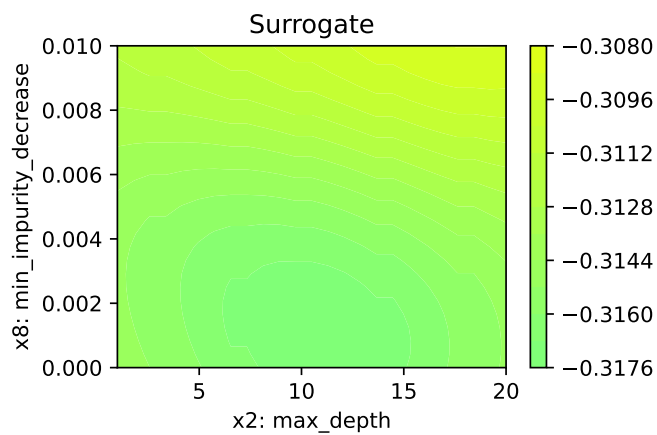
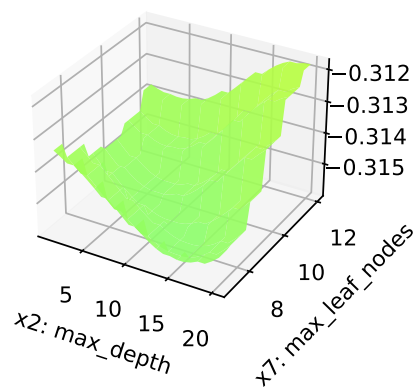
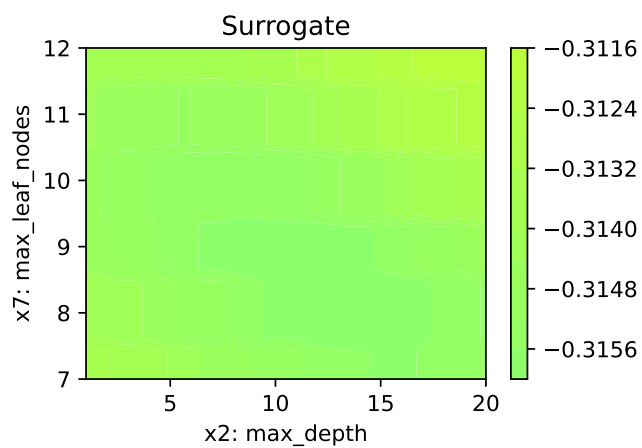
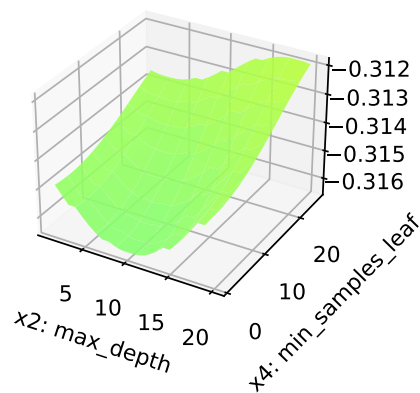
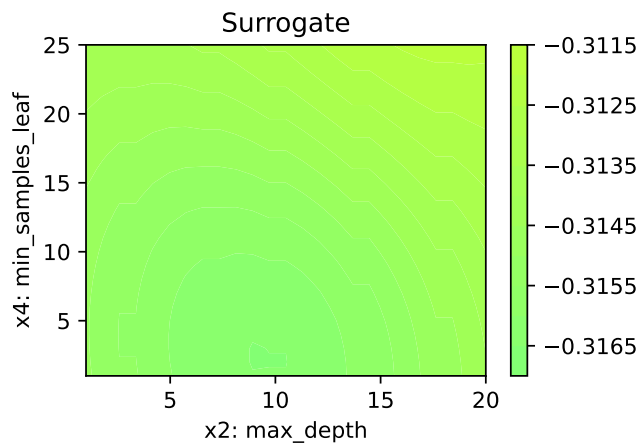
```

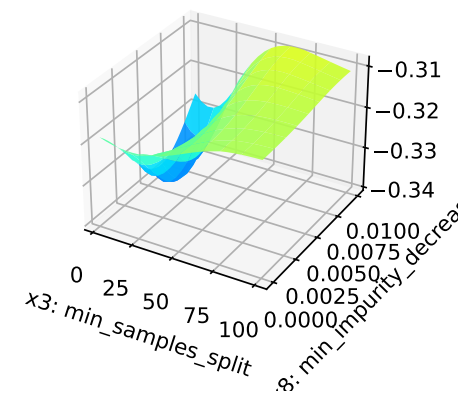
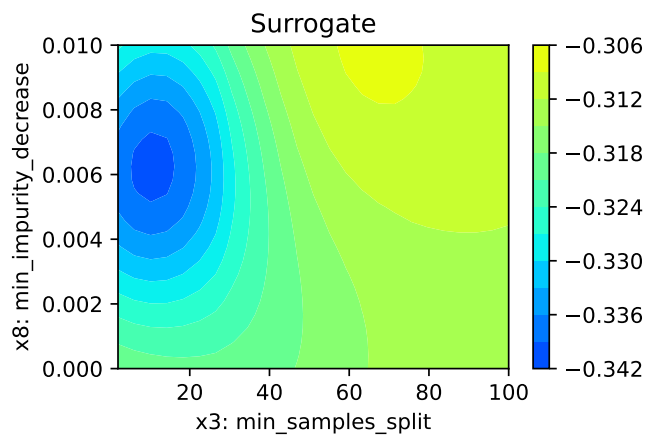
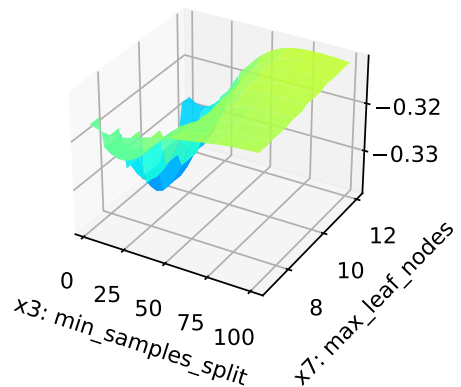
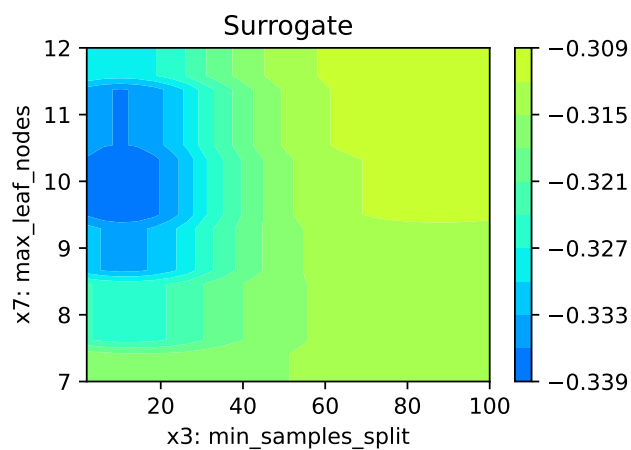
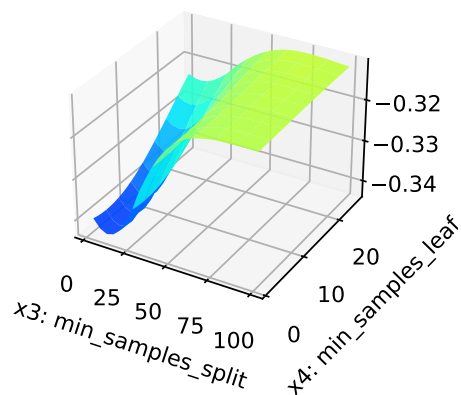
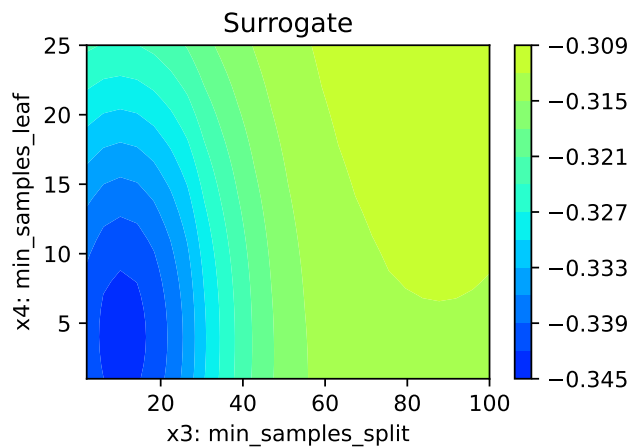



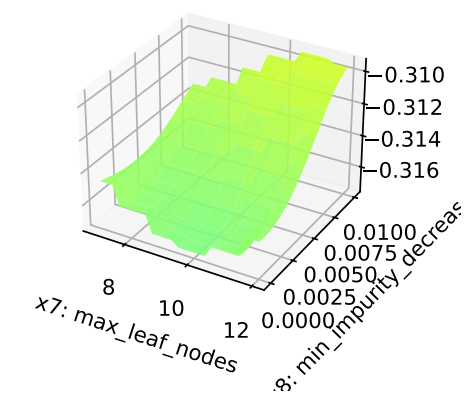
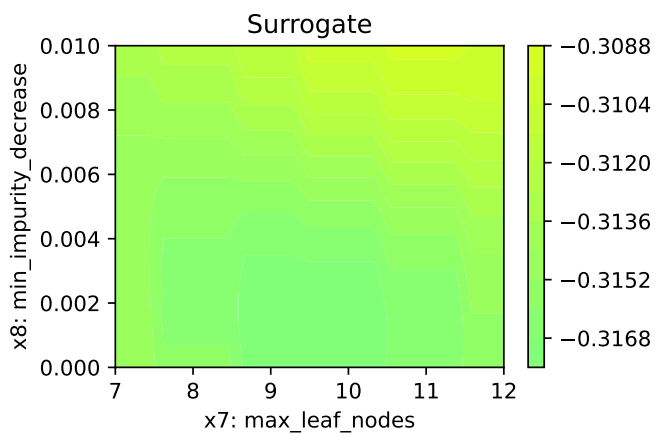
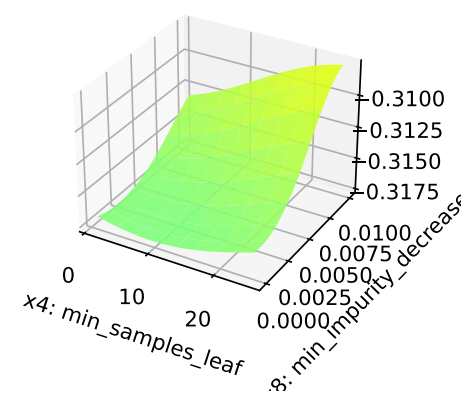
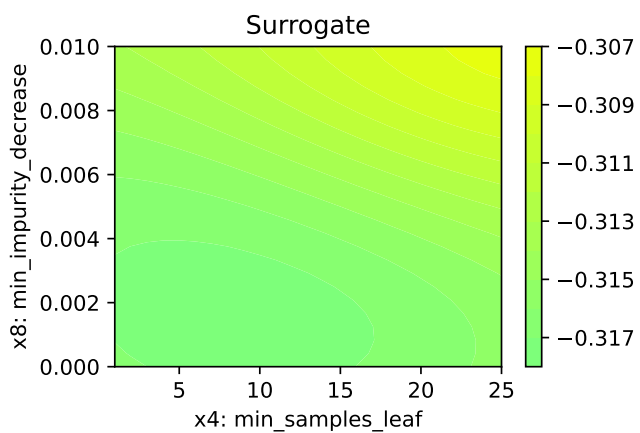
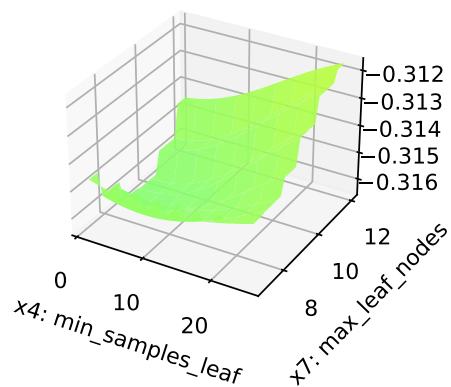
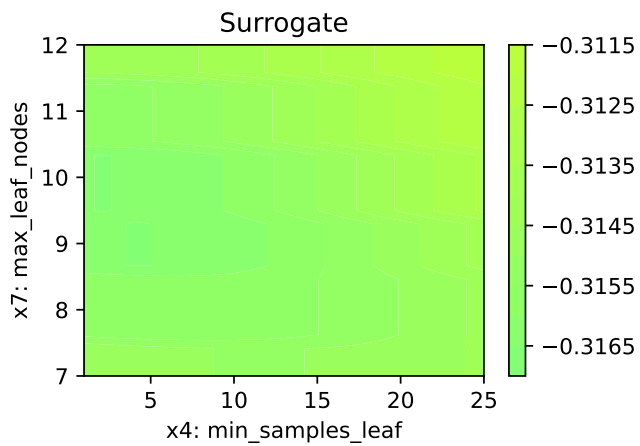












15.10.10 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): text/html

15.10.11 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

```
PLOT_ALL = False
if PLOT_ALL:
    n = spot_tuner.k
    for i in range(n-1):
        for j in range(i+1, n):
            spot_tuner.plot_contour(i=i, j=j, min_z=min_z, max_z = max_z)
```


16 HPT: sklearn XGB Classifier VBDP Data

This document refers to the following software versions:

- python: 3.10.10

```
pip list | grep "spot[RiverPython]"
```

spotPython	0.2.52
spotRiver	0.0.94

Note: you may need to restart the kernel to use updated packages.

spotPython can be installed via pip. Alternatively, the source code can be downloaded from gitHub: <https://github.com/sequential-parameter-optimization/spotPython>.

```
!pip install spotPython
```

- Uncomment the following lines if you want to for (re-)installation the latest version of spotPython from gitHub.

```
# import sys
# !{sys.executable} -m pip install --upgrade build
# !{sys.executable} -m pip install --upgrade --force-reinstall spotPython
```

16.1 Step 1: Setup

Before we consider the detailed experimental setup, we select the parameters that affect run time and the initial design size.

```
MAX_TIME = 1
INIT_SIZE = 5
ORIGINAL = False
```

```

import os
import copy
import socket
from datetime import datetime
from dateutil.tz import tzlocal
start_time = datetime.now(tzlocal())
HOSTNAME = socket.gethostname().split(".")[0]
experiment_name = '17-xgb-sklearn' + "_" + HOSTNAME + "_" + str(MAX_TIME) + "min_" + str(I
experiment_name = experiment_name.replace(':', '-')
print(experiment_name)
if not os.path.exists('./figures'):
    os.makedirs('./figures')

```

17-xgb-sklearn_maans03_1min_5init_2023-07-03_12-58-29

```

import warnings
warnings.filterwarnings("ignore")

```

16.2 Step 2: Initialization of the Empty fun_control Dictionary

 Caution: Tensorboard does not work under Windows

- Since tensorboard does not work under Windows, we recommend setting the parameter `tensorboard_path` to `None` if you are working under Windows.

```

from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="classification",
    tensorboard_path="runs/16_spot_hpt_sklearn_classification")

```

16.3 Step 3: PyTorch Data Loading

16.3.1 1. Load Data: Classification VBDP

```
import pandas as pd
if ORIGINAL == True:
    train_df = pd.read_csv('./data/VBDP/trainnn.csv')
    test_df = pd.read_csv('./data/VBDP/testtt.csv')
else:
    train_df = pd.read_csv('./data/VBDP/train.csv')
    # remove the id column
    train_df = train_df.drop(columns=['id'])

from sklearn.preprocessing import OrdinalEncoder
n_samples = train_df.shape[0]
n_features = train_df.shape[1] - 1
target_column = "prognosis"
# Encoder our prognosis labels as integers for easier decoding later
enc = OrdinalEncoder()
train_df[target_column] = enc.fit_transform(train_df[[target_column]])
train_df.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
print(train_df.shape)
train_df.head()
```

(707, 65)

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19
0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0
1	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0
2	0.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0
3	0.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

The full data set `train_df` 64 features. The target column is labeled as `prognosis`.

16.3.2 Holdout Train and Test Data

We split out a hold-out test set (25% of the data) so we can calculate an example MAP@K

```

import numpy as np
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(train_df.drop(target_column, axis=1),
                                                    random_state=42,
                                                    test_size=0.25,
                                                    stratify=train_df[target_column])

train = pd.DataFrame(np.hstack((X_train, np.array(y_train).reshape(-1, 1))))
test = pd.DataFrame(np.hstack((X_test, np.array(y_test).reshape(-1, 1))))
train.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
test.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
print(train.shape)
print(test.shape)
train.head()

```

(530, 65)

(177, 65)

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0
2	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	1.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0
4	0.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0

```

# add the dataset to the fun_control
fun_control.update({"data": train_df, # full dataset,
                  "train": train,
                  "test": test,
                  "n_samples": n_samples,
                  "target_column": target_column})

```

16.4 Step 4: Specification of the Preprocessing Model

Data preprocesssing can be very simple, e.g., you can ignore it. Then you would choose the `prep_model` “None”:

```

prep_model = None
fun_control.update({"prep_model": prep_model})

```

A default approach for numerical data is the `StandardScaler` (mean 0, variance 1). This can be selected as follows:

```
# prep_model = StandardScaler()
# fun_control.update({"prep_model": prep_model})
```

Even more complicated pre-processing steps are possible, e.g., the following pipeline:

```
# categorical_columns = []
# one_hot_encoder = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
# prep_model = ColumnTransformer(
#     transformers=[
#         ("categorical", one_hot_encoder, categorical_columns),
#     ],
#     remainder=StandardScaler(),
# )
```

16.5 Step 5: Select Model (algorithm) and `core_model_hyper_dict`

The selection of the algorithm (ML model) that should be tuned is done by specifying the its name from the `sklearn` implementation. For example, the SVC support vector machine classifier is selected as follows:

```
fun_control = add_core_model_to_fun_control(SVC, fun_control, SklearnHyperDict)
```

Other `core_models` are, e.g.,:

- `RidgeCV`
- `GradientBoostingRegressor`
- `ElasticNet`
- `RandomForestClassifier`
- `LogisticRegression`
- `KNeighborsClassifier`
- `RandomForestClassifier`
- `GradientBoostingClassifier`
- `HistGradientBoostingClassifier`

We will use the `RandomForestClassifier` classifier in this example.

```

from sklearn.linear_model import RidgeCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.linear_model import ElasticNet
from spotPython.hyperparameters.values import add_core_model_to_fun_control
from spotPython.data.sklearn_hyper_dict import SklearnHyperDict
from spotPython.fun.hypersklearn import HyperSklearn

```

```

# core_model = RidgeCV
# core_model = GradientBoostingRegressor
# core_model = ElasticNet
core_model = RandomForestClassifier
# core_model = SVC
# core_model = LogisticRegression
# core_model = KNeighborsClassifier
# core_model = GradientBoostingClassifier
core_model = HistGradientBoostingClassifier
fun_control = add_core_model_to_fun_control(core_model=core_model,
                                           fun_control=fun_control,
                                           hyper_dict=SklearnHyperDict,
                                           filename=None)

```

Now `fun_control` has the information from the JSON file. The available hyperparameters are:

```

print(*fun_control["core_model_hyper_dict"].keys(), sep="\n")

```

```

loss
learning_rate
max_iter
max_leaf_nodes
max_depth
min_samples_leaf
l2_regularization
max_bins
early_stopping

```

```
n_iter_no_change
tol
```

16.6 Step 6: Modify hyper_dict Hyperparameters for the Selected Algorithm aka core_model

16.6.1 Modify hyperparameter of type numeric and integer (boolean)

Numeric and boolean values can be modified using the `modify_hyper_parameter_bounds` method. For example, to change the `tol` hyperparameter of the SVC model to the interval `[1e-3, 1e-2]`, the following code can be used:

```
fun_control = modify_hyper_parameter_bounds(fun_control, "tol", bounds=[1e-3,
1e-2])
```

```
from spotPython.hyperparameters.values import modify_hyper_parameter_bounds
# fun_control = modify_hyper_parameter_bounds(fun_control, "tol", bounds=[1e-3, 1e-2])
# fun_control = modify_hyper_parameter_bounds(fun_control, "min_samples_split", bounds=[3,
# fun_control = modify_hyper_parameter_bounds(fun_control, "dual", bounds=[0, 0])
# fun_control = modify_hyper_parameter_bounds(fun_control, "probability", bounds=[1, 1])
# fun_control["core_model_hyper_dict"]["tol"]
# fun_control = modify_hyper_parameter_bounds(fun_control, "min_samples_leaf", bounds=[1,
# fun_control = modify_hyper_parameter_bounds(fun_control, "n_estimators", bounds=[5, 10])
```

16.6.2 Modify hyperparameter of type factor

`spotPython` provides functions for modifying the hyperparameters, their bounds and factors as well as for activating and de-activating hyperparameters without re-compilation of the Python source code. These functions were described in [Section 14.6](#).

Factors can be modified with the `modify_hyper_parameter_levels` function. For example, to exclude the `sigmoid` kernel from the tuning, the `kernel` hyperparameter of the SVC model can be modified as follows:

```
fun_control = modify_hyper_parameter_levels(fun_control, "kernel", ["linear",
"rbf"])
```

The new setting can be controlled via:

```
fun_control["core_model_hyper_dict"]["kernel"]
```

```
from spotPython.hyperparameters.values import modify_hyper_parameter_levels
# XGBoost:
fun_control = modify_hyper_parameter_levels(fun_control, "loss", ["log_loss"])
```

16.6.3 Optimizers

Optimizers are described in Section [14.6.1](#).

16.7 Step 7: Selection of the Objective (Loss) Function

16.7.1 Evaluation

The evaluation procedure requires the specification of two elements:

1. the way how the data is split into a train and a test set and
2. the loss function (and a metric).

16.7.2 Selection of the Objective: Metric and Loss Functions

- Machine learning models are optimized with respect to a metric, for example, the accuracy function.
- Deep learning, e.g., neural networks are optimized with respect to a loss function, for example, the `cross_entropy` function and evaluated with respect to a metric, for example, the accuracy function.

16.7.3 Loss Function

The loss function, that is usually used in deep learning for optimizing the weights of the net, is stored in the `fun_control` dictionary as `"loss_function"`.

16.7.4 Metric Function

There are two different types of metrics in `spotPython`:

1. `"metric_river"` is used for the river based evaluation via `eval_oml_iter_progressive`.
2. `"metric_sklearn"` is used for the sklearn based evaluation.

We will consider multi-class classification metrics, e.g., `mapk_score` and `top_k_accuracy_score`.

i Predict Probabilities

In this multi-class classification example the machine learning algorithm should return the probabilities of the specific classes ("**predict_proba**") instead of the predicted values.

We set "**predict_proba**" to **True** in the **fun_control** dictionary.

16.7.4.1 The MAPK Metric

To select the MAPK metric, the following two entries can be added to the **fun_control** dictionary:

```
"metric_sklearn": mapk_score"  
"metric_params": {"k": 3}.
```

16.7.4.2 Other Metrics

Alternatively, other metrics for multi-class classification can be used, e.g., * **top_k_accuracy_score** or * **roc_auc_score**

The metric **roc_auc_score** requires the parameter "**multi_class**", e.g.,

```
"multi_class": "ovr".
```

This is set in the **fun_control** dictionary.

i Weights

spotPython performs a minimization, therefore, metrics that should be maximized have to be multiplied by -1. This is done by setting "**weights**" to -1.

- The complete setup for the metric in our example is:

```
from spotPython.utils.metrics import mapk_score  
fun_control.update({  
    "weights": -1,  
    "metric_sklearn": mapk_score,  
    "predict_proba": True,  
    "metric_params": {"k": 3},  
})
```

16.7.5 Evaluation on Hold-out Data

- The default method for computing the performance is "eval_holdout".
- Alternatively, cross-validation can be used for every machine learning model.
- Specifically for RandomForests, the OOB-score can be used.

```
fun_control.update({
    "eval": "train_hold_out",
})
```

16.7.5.1 Cross Validation

Instead of using the OOB-score, the classical cross validation can be used. The number of folds is set by the key "k_folds". For example, to use 5-fold cross validation, the key "k_folds" is set to 5. Uncomment the following line to use cross validation:

```
# fun_control.update({
#     "eval": "train_cv",
#     "k_folds": 10,
# })
```

16.8 Step 8: Calling the SPOT Function

16.8.1 Preparing the SPOT Call

- Get types and variable names as well as lower and upper bounds for the hyperparameters.

```
# extract the variable types, names, and bounds
from spotPython.hyperparameters.values import (get_bound_values,
    get_var_name,
    get_var_type,)
var_type = get_var_type(fun_control)
var_name = get_var_name(fun_control)
fun_control.update({"var_type": var_type,
    "var_name": var_name})
lower = get_bound_values(fun_control, "lower")
upper = get_bound_values(fun_control, "upper")
```

```
from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control))
```

name	type	default	lower	upper	transform
loss	factor	log_loss	0	0	None
learning_rate	float	-1.0	-5	0	transform_power_10
max_iter	int	7	3	10	transform_power_2_int
max_leaf_nodes	int	5	1	12	transform_power_2_int
max_depth	int	2	1	20	transform_power_2_int
min_samples_leaf	int	4	2	10	transform_power_2_int
l2_regularization	float	0.0	0	10	None
max_bins	int	255	127	255	None
early_stopping	factor	1	0	1	None
n_iter_no_change	int	10	5	20	None
tol	float	0.0001	1e-05	0.001	None

16.8.2 The Objective Function

The objective function is selected next. It implements an interface from `sklearn`'s training, validation, and testing methods to `spotPython`.

```
from spotPython.fun.hypersklearn import HyperSklearn
fun = HyperSklearn().fun_sklearn
```

16.8.3 Run the Spot Optimizer

- Run SPOT for approx. x mins (`max_time`).
- Note: the run takes longer, because the evaluation time of initial design (here: `init_size`, 20 points) is not considered.

```
from spotPython.hyperparameters.values import get_default_hyperparameters_as_array
hyper_dict=SklearnHyperDict().load()
X_start = get_default_hyperparameters_as_array(fun_control, hyper_dict)
X_start
```

```
array([[ 0.00e+00, -1.00e+00,  7.00e+00,  5.00e+00,  2.00e+00,  4.00e+00,
         0.00e+00,  2.55e+02,  1.00e+00,  1.00e+01,  1.00e-04]])
```

```

import numpy as np
from spotPython.spot import spot
from math import inf
spot_tuner = spot.Spot(fun=fun,
                        lower = lower,
                        upper = upper,
                        fun_evals = inf,
                        fun_repeats = 1,
                        max_time = MAX_TIME,
                        noise = False,
                        tolerance_x = np.sqrt(np.spacing(1)),
                        var_type = var_type,
                        var_name = var_name,
                        infill_criterion = "y",
                        n_points = 1,
                        seed=123,
                        log_level = 50,
                        show_models= False,
                        show_progress= True,
                        fun_control = fun_control,
                        design_control={"init_size": INIT_SIZE,
                                      "repeats": 1},
                        surrogate_control={"noise": True,
                                          "cod_type": "norm",
                                          "min_theta": -4,
                                          "max_theta": 3,
                                          "n_theta": len(var_name),
                                          "model_fun_evals": 10_000,
                                          "log_level": 50
                                          })

spot_tuner.run(X_start=X_start)

```

spotPython tuning: -0.38847117794486213 [-----] 3.71%

spotPython tuning: -0.38847117794486213 [#-----] 6.29%

spotPython tuning: -0.38847117794486213 [#-----] 9.49%

spotPython tuning: -0.38847117794486213 [#-----] 12.34%

spotPython tuning: -0.38847117794486213 [##-----] 16.96%

```

spotPython tuning: -0.38847117794486213 [##-----] 21.70%
spotPython tuning: -0.38847117794486213 [###-----] 25.05%
spotPython tuning: -0.38847117794486213 [###-----] 28.21%
spotPython tuning: -0.38847117794486213 [###-----] 31.57%
spotPython tuning: -0.38847117794486213 [####-----] 36.37%
spotPython tuning: -0.38847117794486213 [####-----] 39.04%
spotPython tuning: -0.38847117794486213 [#####-----] 46.41%
spotPython tuning: -0.38847117794486213 [#####-----] 52.07%
spotPython tuning: -0.38847117794486213 [#####-----] 58.86%
spotPython tuning: -0.38847117794486213 [#####-----] 68.03%
spotPython tuning: -0.38847117794486213 [#####-----] 74.16%
spotPython tuning: -0.38847117794486213 [#####-----] 85.88%
spotPython tuning: -0.38847117794486213 [#####-----] 94.55%
spotPython tuning: -0.38847117794486213 [#####-----] 100.00% Done...

<spotPython.spot.spot.Spot at 0x17fd8f700>

```

16.9 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section 14.9, see also the description in the documentation: [Tensorboard](#).

16.10 Step 10: Results

After the hyperparameter tuning run is finished, the progress of the hyperparameter tuning can be visualized. The following code generates the progress plot from `?@fig-progress`.

```
spot_tuner.plot_progress(log_y=False,
                        filename="./figures/" + experiment_name+"_progress.png")
```

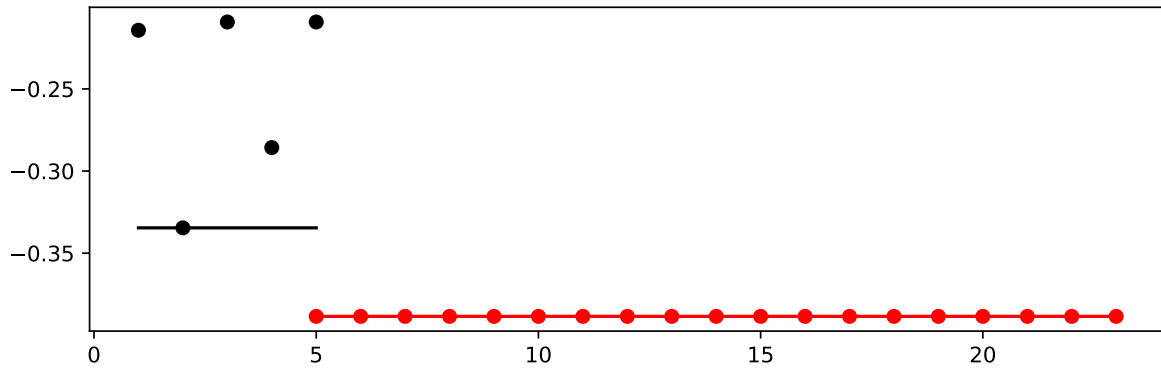


Figure 16.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- Print the results

```
print(gen_design_table(fun_control=fun_control,
                      spot=spot_tuner))
```

name	type	default	lower	upper	tuned	trans
loss	factor	log_loss	0.0	0.0	0.0	None
learning_rate	float	-1.0	-5.0	0.0	-0.9302847174013998	trans
max_iter	int	7	3.0	10.0	9.0	trans
max_leaf_nodes	int	5	1.0	12.0	5.0	trans
max_depth	int	2	1.0	20.0	19.0	trans
min_samples_leaf	int	4	2.0	10.0	2.0	trans
l2_regularization	float	0.0	0.0	10.0	2.402908317415685	None
max_bins	int	255	127.0	255.0	142.0	None
early_stopping	factor	1	0.0	1.0	1.0	None
n_iter_no_change	int	10	5.0	20.0	6.0	None
tol	float	0.0001	1e-05	0.001	0.0009512858283515322	None

16.10.1 Show variable importance

```
spot_tuner.plot_importance(threshold=0.025, filename="./figures/" + experiment_name+"_impo
```

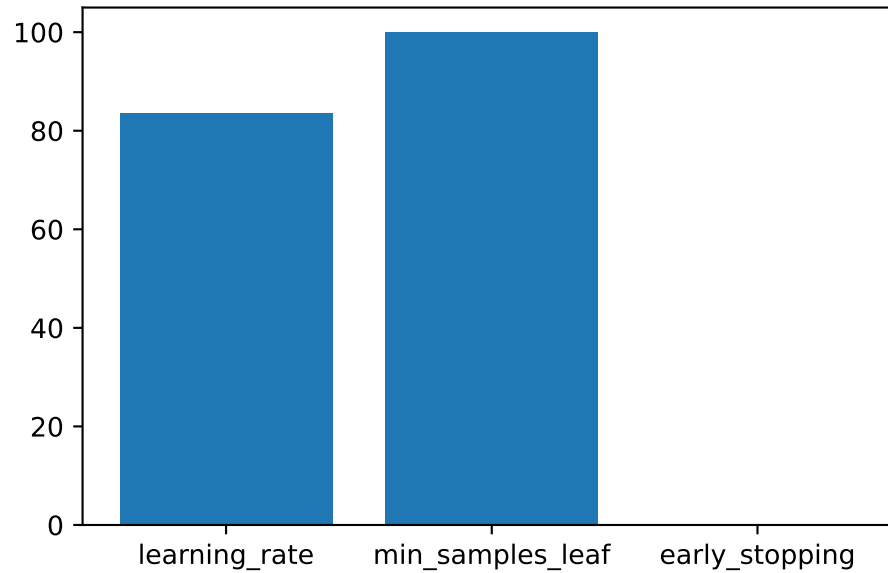


Figure 16.2: Variable importance plot, threshold 0.025.

16.10.2 Get Default Hyperparameters

```
from spotPython.hyperparameters.values import get_default_values, transform_hyper_parameter_values_default = get_default_values(fun_control)
values_default = transform_hyper_parameter_values(fun_control=fun_control, hyper_parameter_values_default
```

```
{'loss': 'log_loss',
 'learning_rate': 0.1,
 'max_iter': 128,
 'max_leaf_nodes': 32,
 'max_depth': 4,
 'min_samples_leaf': 16,
 'l2_regularization': 0.0,
 'max_bins': 255,
 'early_stopping': 1,
```

```
'n_iter_no_change': 10,
'tol': 0.0001}
```

```
from sklearn.pipeline import make_pipeline
model_default = make_pipeline(fun_control["prep_model"], fun_control["core_model"](**value
model_default
```

```
Pipeline(steps=[('nonetype', None),
                 ('histgradientboostingclassifier',
                  HistGradientBoostingClassifier(early_stopping=1, max_depth=4,
                                                  max_iter=128, max_leaf_nodes=32,
                                                  min_samples_leaf=16,
                                                  tol=0.0001))])
```

16.10.3 Get SPOT Results

```
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
print(X)
```

```
[[ 0.00000000e+00 -9.30284717e-01  9.00000000e+00  5.00000000e+00
  1.90000000e+01  2.00000000e+00  2.40290832e+00  1.42000000e+02
  1.00000000e+00  6.00000000e+00  9.51285828e-04]]
```

```
from spotPython.hyperparameters.values import assign_values, return_conf_list_from_var_dict
v_dict = assign_values(X, fun_control["var_name"])
return_conf_list_from_var_dict(var_dict=v_dict, fun_control=fun_control)
```

```
[{'loss': 'log_loss',
 'learning_rate': 0.11741275609254138,
 'max_iter': 512,
 'max_leaf_nodes': 32,
 'max_depth': 524288,
 'min_samples_leaf': 4,
 'l2_regularization': 2.402908317415685,
 'max_bins': 142,
 'early_stopping': 1,
 'n_iter_no_change': 6,
 'tol': 0.0009512858283515322}]
```



```

from spotPython.hyperparameters.values import get_one_sklearn_model_from_X
model_spot = get_one_sklearn_model_from_X(X, fun_control)
model_spot

```

```

HistGradientBoostingClassifier(early_stopping=1,
                                l2_regularization=2.402908317415685,
                                learning_rate=0.11741275609254138, max_bins=142,
                                max_depth=524288, max_iter=512,
                                max_leaf_nodes=32, min_samples_leaf=4,
                                n_iter_no_change=6, tol=0.0009512858283515322)

```

16.10.4 Evaluate SPOT Results

- Fetch the data.

```

from spotPython.utils.convert import get_Xy_from_df
X_train, y_train = get_Xy_from_df(fun_control["train"], fun_control["target_column"])
X_test, y_test = get_Xy_from_df(fun_control["test"], fun_control["target_column"])
X_test.shape, y_test.shape

```

```
((177, 64), (177,))
```

- Fit the model with the tuned hyperparameters. This gives one result:

```

model_spot.fit(X_train, y_train)
y_pred = model_spot.predict_proba(X_test)
res = mapk_score(y_true=y_test, y_pred=y_pred, k=3)
res

```

```
0.3549905838041431
```

```

def repeated_eval(n, model):
    res_values = []
    for i in range(n):
        model.fit(X_train, y_train)
        y_pred = model.predict_proba(X_test)
        res = mapk_score(y_true=y_test, y_pred=y_pred, k=3)
        res_values.append(res)
    mean_res = np.mean(res_values)

```

```

print(f"mean_res: {mean_res}")
std_res = np.std(res_values)
print(f"std_res: {std_res}")
min_res = np.min(res_values)
print(f"min_res: {min_res}")
max_res = np.max(res_values)
print(f"max_res: {max_res}")
median_res = np.median(res_values)
print(f"median_res: {median_res}")
return mean_res, std_res, min_res, max_res, median_res

```

16.10.5 Handling Non-deterministic Results

- Because the model is non-deterministic, we perform $n = 30$ runs and calculate the mean and standard deviation of the performance metric.

```
_ = repeated_eval(30, model_spot)
```

```

mean_res: 0.3413998744507218
std_res: 0.015250523315698395
min_res: 0.30414312617702444
max_res: 0.37099811676082856
median_res: 0.3422787193973635

```

16.10.6 Evaluation of the Default Hyperparameters

```
model_default.fit(X_train, y_train)["histgradientboostingclassifier"]
```

```

HistGradientBoostingClassifier(early_stopping=1, max_depth=4, max_iter=128,
                                max_leaf_nodes=32, min_samples_leaf=16,
                                tol=0.0001)

```

- One evaluation of the default hyperparameters is performed on the hold-out test set.

```

y_pred = model_default.predict_proba(X_test)
mapk_score(y_true=y_test, y_pred=y_pred, k=3)

```

```
0.3427495291902071
```

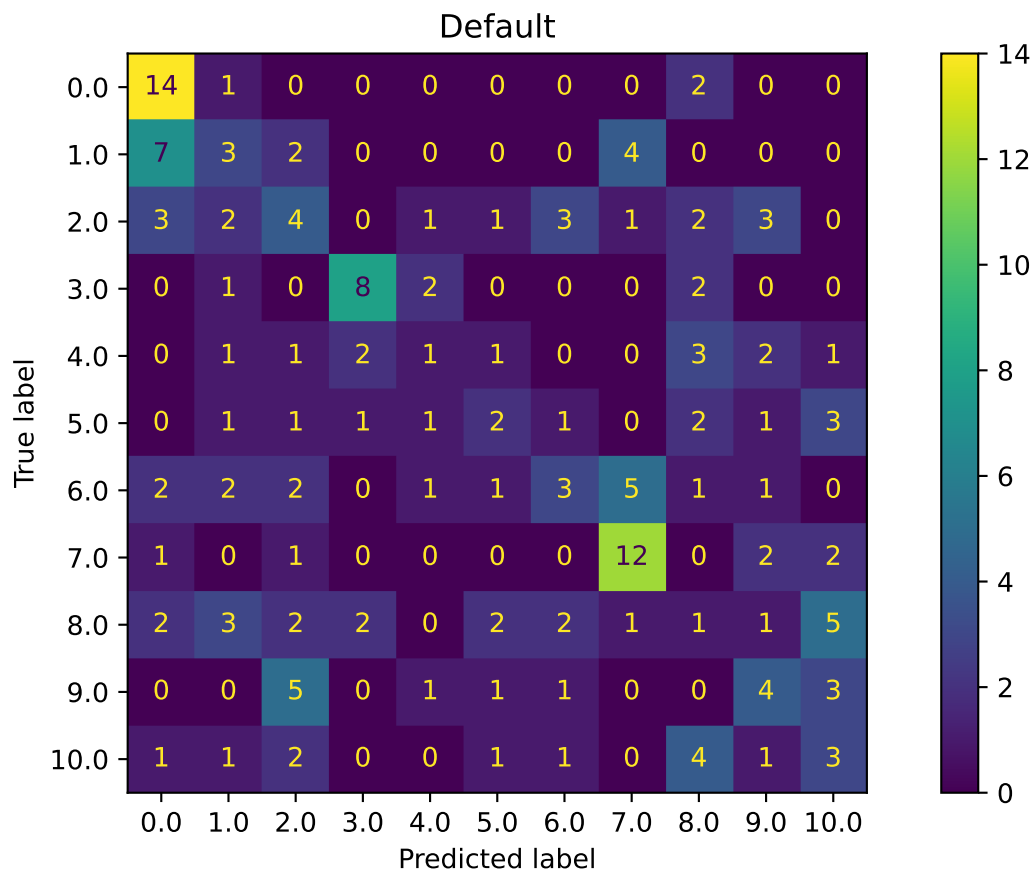
Since one single evaluation is not meaningful, we perform, similar to the evaluation of the SPOT results, $n = 30$ runs of the default setting and calculate the mean and standard deviation of the performance metric.

```
_ = repeated_eval(30, model_default)
```

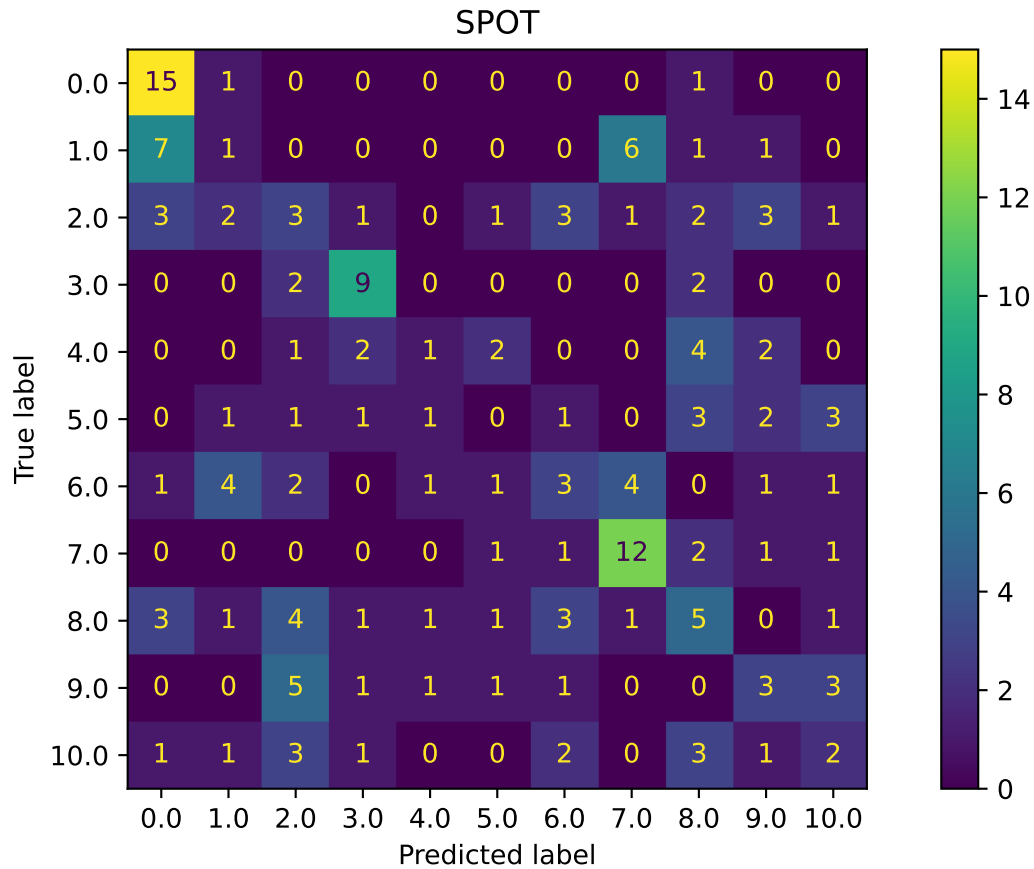
```
mean_res: 0.34978028876333955
std_res: 0.015339506806159094
min_res: 0.3163841807909605
max_res: 0.38229755178907715
median_res: 0.3516949152542373
```

16.10.7 Plot: Compare Predictions

```
from spotPython.plot.validation import plot_confusion_matrix
plot_confusion_matrix(model_default, fun_control, title = "Default")
```



```
plot_confusion_matrix(model_spot, fun_control, title="SPOT")
```



```
min(spot_tuner.y), max(spot_tuner.y)
```

```
(-0.38847117794486213, -0.20927318295739344)
```

16.10.8 Cross-validated Evaluations

```
from spotPython.sklearn.traintest import evaluate_cv
fun_control.update({
    "eval": "train_cv",
    "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)
```

```
(0.31949685534591193, None)
```

```

fun_control.update({
    "eval": "test_cv",
    "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)

```

(0.27325708061002174, None)

- This is the evaluation that will be used in the comparison:

```

fun_control.update({
    "eval": "data_cv",
    "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)

```

(0.3450503018108652, None)

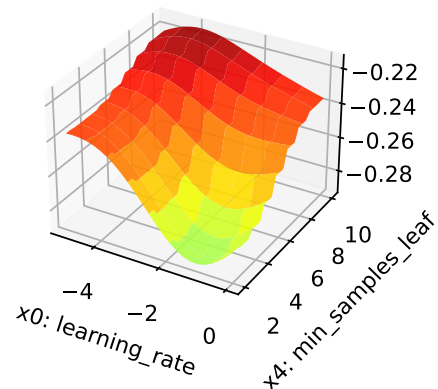
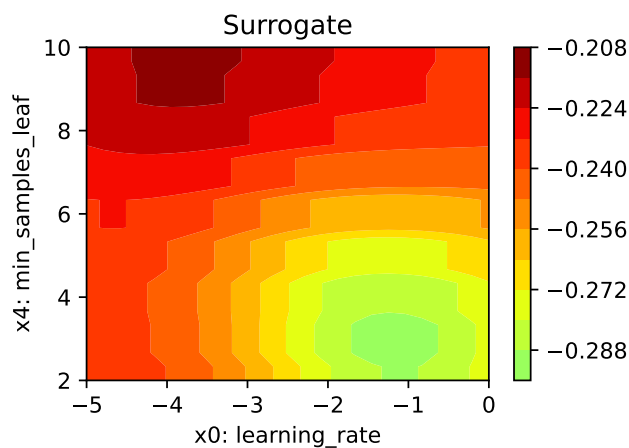
16.10.9 Detailed Hyperparameter Plots

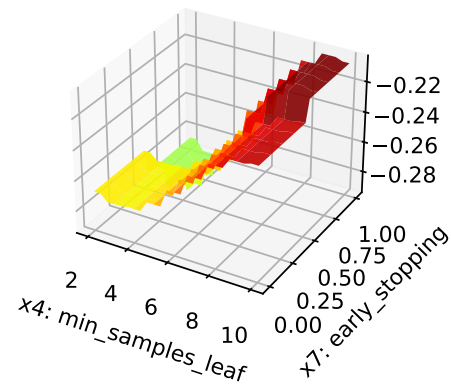
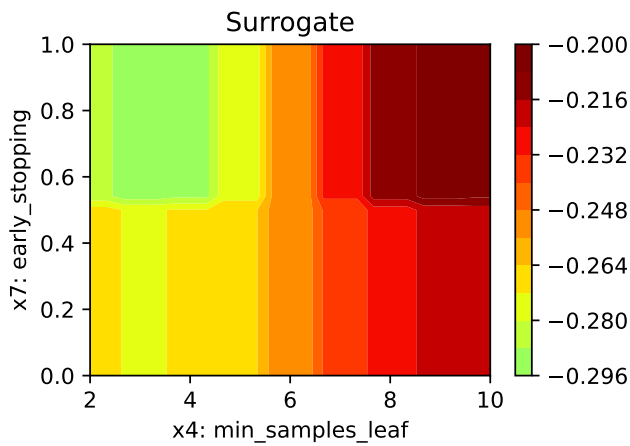
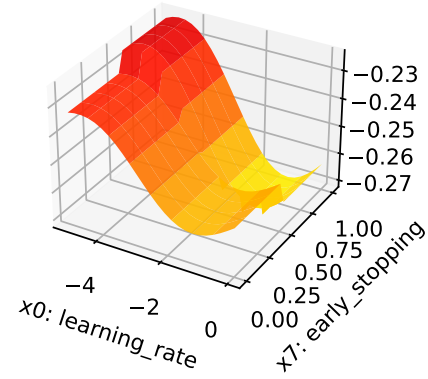
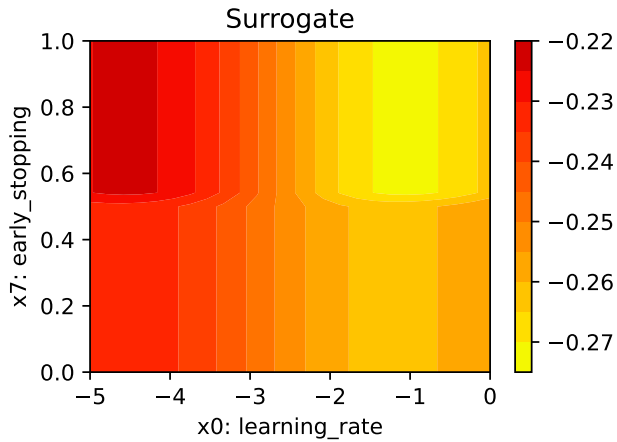
```

filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)

```

learning_rate: 83.54529818571723
min_samples_leaf: 100.0
early_stopping: 0.058100601132847866





16.10.10 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): text/html

16.10.11 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

```
PLOT_ALL = False
if PLOT_ALL:
    n = spot_tuner.k
    for i in range(n-1):
        for j in range(i+1, n):
            spot_tuner.plot_contour(i=i, j=j, min_z=min_z, max_z = max_z)
```


17 HPT: sklearn SVC VBDP Data

This document refers to the following software versions:

- python: 3.10.10

```
pip list | grep "spot[RiverPython]"
```

spotPython	0.2.52
spotRiver	0.0.94

Note: you may need to restart the kernel to use updated packages.

spotPython can be installed via pip. Alternatively, the source code can be downloaded from gitHub: <https://github.com/sequential-parameter-optimization/spotPython>.

```
!pip install spotPython
```

- Uncomment the following lines if you want to for (re-)installation the latest version of spotPython from gitHub.

```
# import sys
# !{sys.executable} -m pip install --upgrade build
# !{sys.executable} -m pip install --upgrade --force-reinstall spotPython
```

17.1 Step 1: Setup

Before we consider the detailed experimental setup, we select the parameters that affect run time and the initial design size.

```
MAX_TIME = 1
INIT_SIZE = 5
ORIGINAL = False
```

```

import os
import copy
import socket
from datetime import datetime
from dateutil.tz import tzlocal
start_time = datetime.now(tzlocal())
HOSTNAME = socket.gethostname().split(".")[0]
experiment_name = '18-svc-sklearn' + "_" + HOSTNAME + "_" + str(MAX_TIME) + "min_" + str(I
experiment_name = experiment_name.replace(':', '-')
print(experiment_name)
if not os.path.exists('./figures'):
    os.makedirs('./figures')

```

18-svc-sklearn_maans03_1min_5init_2023-07-03_13-02-37

```

import warnings
warnings.filterwarnings("ignore")

```

17.2 Step 2: Initialization of the Empty fun_control Dictionary

 Caution: Tensorboard does not work under Windows

- Since tensorboard does not work under Windows, we recommend setting the parameter `tensorboard_path` to `None` if you are working under Windows.

```

from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="classification",
    tensorboard_path="runs/16_spot_hpt_sklearn_classification")

```

17.3 Step 3: PyTorch Data Loading

17.3.1 1. Load Data: Classification VBDP

```
import pandas as pd
if ORIGINAL == True:
    train_df = pd.read_csv('./data/VBDP/trainnn.csv')
    test_df = pd.read_csv('./data/VBDP/testtt.csv')
else:
    train_df = pd.read_csv('./data/VBDP/train.csv')
    # remove the id column
    train_df = train_df.drop(columns=['id'])

from sklearn.preprocessing import OrdinalEncoder
n_samples = train_df.shape[0]
n_features = train_df.shape[1] - 1
target_column = "prognosis"
# Encoder our prognosis labels as integers for easier decoding later
enc = OrdinalEncoder()
train_df[target_column] = enc.fit_transform(train_df[[target_column]])
train_df.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
print(train_df.shape)
train_df.head()
```

(707, 65)

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19
0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0
1	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0
2	0.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0
3	0.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

The full data set `train_df` 64 features. The target column is labeled as `prognosis`.

17.3.2 Holdout Train and Test Data

We split out a hold-out test set (25% of the data) so we can calculate an example MAP@K

```

import numpy as np
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(train_df.drop(target_column, axis=1),
                                                    random_state=42,
                                                    test_size=0.25,
                                                    stratify=train_df[target_column])

train = pd.DataFrame(np.hstack((X_train, np.array(y_train).reshape(-1, 1))))
test = pd.DataFrame(np.hstack((X_test, np.array(y_test).reshape(-1, 1))))
train.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
test.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
print(train.shape)
print(test.shape)
train.head()

```

(530, 65)

(177, 65)

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0
2	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	1.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0
4	0.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0

```

# add the dataset to the fun_control
fun_control.update({"data": train_df, # full dataset,
                  "train": train,
                  "test": test,
                  "n_samples": n_samples,
                  "target_column": target_column})

```

17.4 Step 4: Specification of the Preprocessing Model

Data preprocesssing can be very simple, e.g., you can ignore it. Then you would choose the `prep_model` “None”:

```

prep_model = None
fun_control.update({"prep_model": prep_model})

```

A default approach for numerical data is the `StandardScaler` (mean 0, variance 1). This can be selected as follows:

```
# prep_model = StandardScaler()
# fun_control.update({"prep_model": prep_model})
```

Even more complicated pre-processing steps are possible, e.g., the following pipeline:

```
# categorical_columns = []
# one_hot_encoder = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
# prep_model = ColumnTransformer(
#     transformers=[
#         ("categorical", one_hot_encoder, categorical_columns),
#     ],
#     remainder=StandardScaler(),
# )
```

17.5 Step 5: Select Model (algorithm) and `core_model_hyper_dict`

The selection of the algorithm (ML model) that should be tuned is done by specifying the its name from the `sklearn` implementation. For example, the SVC support vector machine classifier is selected as follows:

```
fun_control = add_core_model_to_fun_control(SVC, fun_control, SklearnHyperDict)
```

Other `core_models` are, e.g.,:

- `RidgeCV`
- `GradientBoostingRegressor`
- `ElasticNet`
- `RandomForestClassifier`
- `LogisticRegression`
- `KNeighborsClassifier`
- `RandomForestClassifier`
- `GradientBoostingClassifier`
- `HistGradientBoostingClassifier`

We will use the `RandomForestClassifier` classifier in this example.

```

from sklearn.linear_model import RidgeCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.linear_model import ElasticNet
from spotPython.hyperparameters.values import add_core_model_to_fun_control
from spotPython.data.sklearn_hyper_dict import SklearnHyperDict
from spotPython.fun.hypersklearn import HyperSklearn

```

```

# core_model = RidgeCV
# core_model = GradientBoostingRegressor
# core_model = ElasticNet
# core_model = RandomForestClassifier
core_model = SVC
# core_model = LogisticRegression
# core_model = KNeighborsClassifier
# core_model = GradientBoostingClassifier
# core_model = HistGradientBoostingClassifier
fun_control = add_core_model_to_fun_control(core_model=core_model,
                                           fun_control=fun_control,
                                           hyper_dict=SklearnHyperDict,
                                           filename=None)

```

Now `fun_control` has the information from the JSON file. The available hyperparameters are:

```

print(*fun_control["core_model_hyper_dict"].keys(), sep="\n")

```

```

C
kernel
degree
gamma
coef0
shrinking
probability
tol
cache_size

```

break_ties

17.6 Step 6: Modify hyper_dict Hyperparameters for the Selected Algorithm aka core_model

17.6.1 Modify hyperparameter of type numeric and integer (boolean)

Numeric and boolean values can be modified using the `modify_hyper_parameter_bounds` method. For example, to change the `tol` hyperparameter of the SVC model to the interval `[1e-3, 1e-2]`, the following code can be used:

```
fun_control = modify_hyper_parameter_bounds(fun_control, "tol", bounds=[1e-3, 1e-2])
```

```
from spotPython.hyperparameters.values import modify_hyper_parameter_bounds
fun_control = modify_hyper_parameter_bounds(fun_control, "probability", bounds=[1, 1])
```

17.6.2 Modify hyperparameter of type factor

`spotPython` provides functions for modifying the hyperparameters, their bounds and factors as well as for activating and de-activating hyperparameters without re-compilation of the Python source code. These functions were described in [Section 14.6](#).

Factors can be modified with the `modify_hyper_parameter_levels` function. For example, to exclude the `sigmoid` kernel from the tuning, the `kernel` hyperparameter of the SVC model can be modified as follows:

```
fun_control = modify_hyper_parameter_levels(fun_control, "kernel", ["linear", "rbf"])
```

The new setting can be controlled via:

```
fun_control["core_model_hyper_dict"]["kernel"]
```

```
from spotPython.hyperparameters.values import modify_hyper_parameter_levels
fun_control = modify_hyper_parameter_levels(fun_control, "kernel", ["rbf"])
```

17.6.3 Optimizers

Optimizers are described in [Section 14.6.1](#).

17.6.4 Selection of the Objective: Metric and Loss Functions

- Machine learning models are optimized with respect to a metric, for example, the `accuracy` function.
- Deep learning, e.g., neural networks are optimized with respect to a loss function, for example, the `cross_entropy` function and evaluated with respect to a metric, for example, the `accuracy` function.

17.7 Step 7: Selection of the Objective (Loss) Function

The loss function, that is usually used in deep learning for optimizing the weights of the net, is stored in the `fun_control` dictionary as `"loss_function"`.

17.7.1 Metric Function

There are two different types of metrics in `spotPython`:

1. `"metric_river"` is used for the river based evaluation via `eval_oml_iter_progressive`.
2. `"metric_sklearn"` is used for the sklearn based evaluation.

We will consider multi-class classification metrics, e.g., `mapk_score` and `top_k_accuracy_score`.

Predict Probabilities

In this multi-class classification example the machine learning algorithm should return the probabilities of the specific classes (`"predict_proba"`) instead of the predicted values.

We set `"predict_proba"` to `True` in the `fun_control` dictionary.

17.7.1.1 The MAPK Metric

To select the MAPK metric, the following two entries can be added to the `fun_control` dictionary:

```
"metric_sklearn": mapk_score"
```

```
"metric_params": {"k": 3}.
```


17.7.1.2 Other Metrics

Alternatively, other metrics for multi-class classification can be used, e.g.,: `* top_k_accuracy_score` or `* roc_auc_score`

The metric `roc_auc_score` requires the parameter `"multi_class"`, e.g.,

```
"multi_class": "ovr".
```

This is set in the `fun_control` dictionary.

Weights

`spotPython` performs a minimization, therefore, metrics that should be maximized have to be multiplied by -1. This is done by setting `"weights"` to -1.

- The complete setup for the metric in our example is:

```
from spotPython.utils.metrics import mapk_score
fun_control.update({
    "weights": -1,
    "metric_sklearn": mapk_score,
    "predict_proba": True,
    "metric_params": {"k": 3},
})
```

17.7.2 Evaluation on Hold-out Data

- The default method for computing the performance is `"eval_holdout"`.
- Alternatively, cross-validation can be used for every machine learning model.
- Specifically for `RandomForests`, the OOB-score can be used.

```
fun_control.update({
    "eval": "train_hold_out",
})
```

17.7.2.1 Cross Validation

Instead of using the OOB-score, the classical cross validation can be used. The number of folds is set by the key `"k_folds"`. For example, to use 5-fold cross validation, the key `"k_folds"` is set to 5. Uncomment the following line to use cross validation:

```
# fun_control.update({
#     "eval": "train_cv",
#     "k_folds": 10,
# })
```

17.8 Step 8: Calling the SPOT Function

17.8.1 Preparing the SPOT Call

- Get types and variable names as well as lower and upper bounds for the hyperparameters.

```
# extract the variable types, names, and bounds
from spotPython.hyperparameters.values import (get_bound_values,
        get_var_name,
        get_var_type,)
var_type = get_var_type(fun_control)
var_name = get_var_name(fun_control)
fun_control.update({"var_type": var_type,
                    "var_name": var_name})
lower = get_bound_values(fun_control, "lower")
upper = get_bound_values(fun_control, "upper")

from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control))
```

name	type	default	lower	upper	transform
C	float	1.0	0.1	10	None
kernel	factor	rbf	0	0	None
degree	int	3	3	3	None
gamma	factor	scale	0	1	None
coef0	float	0.0	0	0	None
shrinking	factor	0	0	1	None
probability	factor	0	1	1	None
tol	float	0.001	0.0001	0.01	None
cache_size	float	200.0	100	400	None
break_ties	factor	0	0	1	None

17.8.2 The Objective Function

The objective function is selected next. It implements an interface from `sklearn`'s training, validation, and testing methods to `spotPython`.

```
from spotPython.fun.hypersklearn import HyperSklearn
fun = HyperSklearn().fun_sklearn
```

17.8.3 Run the Spot Optimizer

- Run SPOT for approx. x mins (`max_time`).
- Note: the run takes longer, because the evaluation time of initial design (here: `init_size`, 20 points) is not considered.

```
from spotPython.hyperparameters.values import get_default_hyperparameters_as_array
hyper_dict=SklearnHyperDict().load()
X_start = get_default_hyperparameters_as_array(fun_control, hyper_dict)
X_start
```

```
array([[1.e+00, 2.e+00, 3.e+00, 0.e+00, 0.e+00, 0.e+00, 0.e+00, 1.e-03,
        2.e+02, 0.e+00]])
```

```
import numpy as np
from spotPython.spot import spot
from math import inf
spot_tuner = spot.Spot(fun=fun,
                       lower = lower,
                       upper = upper,
                       fun_evals = inf,
                       fun_repeats = 1,
                       max_time = MAX_TIME,
                       noise = False,
                       tolerance_x = np.sqrt(np.spacing(1)),
                       var_type = var_type,
                       var_name = var_name,
                       infill_criterion = "y",
                       n_points = 1,
                       seed=123,
                       log_level = 50,
                       show_models= False,
```

```

show_progress= True,
fun_control = fun_control,
design_control={"init_size": INIT_SIZE,
               "repeats": 1},
surrogate_control={"noise": True,
                   "cod_type": "norm",
                   "min_theta": -4,
                   "max_theta": 3,
                   "n_theta": len(var_name),
                   "model_fun_evals": 10_000,
                   "log_level": 50
                  })

spot_tuner.run(X_start=X_start)

```

```

spotPython tuning: -0.38345864661654133 [-----] 0.79%

spotPython tuning: -0.38345864661654133 [-----] 1.57%

spotPython tuning: -0.38345864661654133 [-----] 3.55%

spotPython tuning: -0.38345864661654133 [-----] 4.68%

spotPython tuning: -0.38345864661654133 [#-----] 5.72%

spotPython tuning: -0.38345864661654133 [#-----] 6.66%

spotPython tuning: -0.38345864661654133 [#-----] 7.57%

spotPython tuning: -0.38345864661654133 [#-----] 8.69%

spotPython tuning: -0.38345864661654133 [#-----] 9.70%

spotPython tuning: -0.38345864661654133 [#-----] 11.03%

spotPython tuning: -0.38345864661654133 [#-----] 12.13%

spotPython tuning: -0.38345864661654133 [#-----] 13.30%

```

spotPython tuning: -0.38345864661654133 [#-----] 14.75%

spotPython tuning: -0.38345864661654133 [##-----] 16.05%

spotPython tuning: -0.38345864661654133 [##-----] 17.40%

spotPython tuning: -0.38345864661654133 [##-----] 18.96%

spotPython tuning: -0.38345864661654133 [##-----] 20.48%

spotPython tuning: -0.38345864661654133 [##-----] 22.05%

spotPython tuning: -0.38345864661654133 [##-----] 23.34%

spotPython tuning: -0.38345864661654133 [##-----] 24.84%

spotPython tuning: -0.38345864661654133 [###-----] 26.44%

spotPython tuning: -0.38345864661654133 [###-----] 28.14%

spotPython tuning: -0.38345864661654133 [###-----] 29.60%

spotPython tuning: -0.38345864661654133 [###-----] 31.21%

spotPython tuning: -0.38345864661654133 [###-----] 32.78%

spotPython tuning: -0.38345864661654133 [###-----] 34.39%

spotPython tuning: -0.38345864661654133 [####-----] 35.98%

spotPython tuning: -0.38345864661654133 [####-----] 37.74%

spotPython tuning: -0.38345864661654133 [####-----] 39.53%

spotPython tuning: -0.38345864661654133 [####-----] 41.47%

spotPython tuning: -0.38345864661654133 [####-----] 43.30%

spotPython tuning: -0.38345864661654133 [#####-----] 45.09%

spotPython tuning: -0.38345864661654133 [#####-----] 46.84%

spotPython tuning: -0.38596491228070173 [#####-----] 48.72%

spotPython tuning: -0.38596491228070173 [#####-----] 50.57%

spotPython tuning: -0.38596491228070173 [#####-----] 52.29%

spotPython tuning: -0.38596491228070173 [#####-----] 54.10%

spotPython tuning: -0.38596491228070173 [#####-----] 55.78%

spotPython tuning: -0.38596491228070173 [#####-----] 57.86%

spotPython tuning: -0.38596491228070173 [#####-----] 59.77%

spotPython tuning: -0.38596491228070173 [#####-----] 61.58%

spotPython tuning: -0.38596491228070173 [#####-----] 63.60%

spotPython tuning: -0.38596491228070173 [#####-----] 65.58%

spotPython tuning: -0.38596491228070173 [#####-----] 67.54%

spotPython tuning: -0.38596491228070173 [#####-----] 69.67%

spotPython tuning: -0.38596491228070173 [#####-----] 71.81%

spotPython tuning: -0.38596491228070173 [#####-----] 73.82%

spotPython tuning: -0.38596491228070173 [#####-----] 75.78%

spotPython tuning: -0.38596491228070173 [#####-----] 77.72%

spotPython tuning: -0.38596491228070173 [#####-----] 79.71%

```

spotPython tuning: -0.38596491228070173 [#####--] 81.69%

spotPython tuning: -0.38596491228070173 [#####--] 84.00%

spotPython tuning: -0.38596491228070173 [#####-] 86.18%

spotPython tuning: -0.38596491228070173 [#####-] 88.40%

spotPython tuning: -0.38596491228070173 [#####-] 90.64%

spotPython tuning: -0.38596491228070173 [#####-] 92.84%

spotPython tuning: -0.38596491228070173 [#####-] 94.81%

spotPython tuning: -0.38596491228070173 [#####] 96.84%

spotPython tuning: -0.38596491228070173 [#####] 100.00% Done...

<spotPython.spot.spot.Spot at 0x180463520>

```

17.9 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section 14.9, see also the description in the documentation: [Tensorboard](#).

17.10 Step 10: Results

After the hyperparameter tuning run is finished, the progress of the hyperparameter tuning can be visualized. The following code generates the progress plot from `?@fig-progress`.

```

spot_tuner.plot_progress(log_y=False,
    filename="./figures/" + experiment_name+"_progress.png")

```

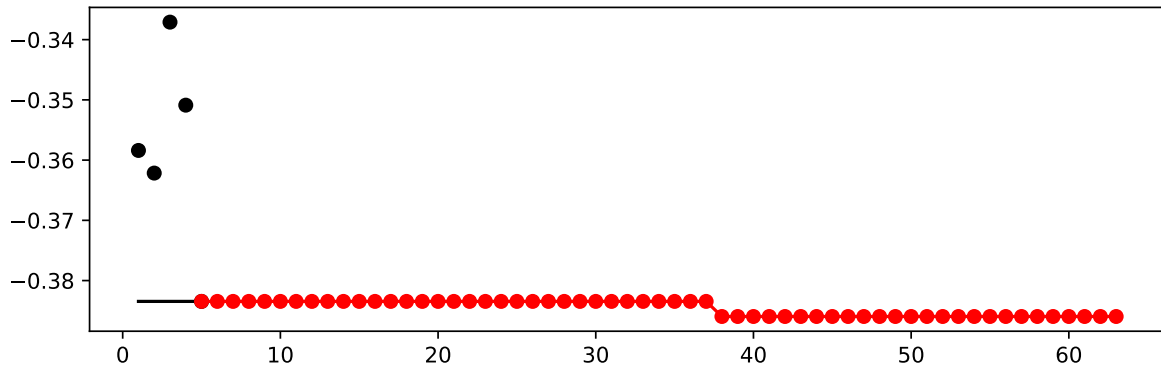


Figure 17.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- Print the results

```
print(gen_design_table(fun_control=fun_control,
                      spot=spot_tuner))
```

name	type	default	lower	upper	tuned	transform
C	float	1.0	0.1	10.0	7.640370712487234	None
kernel	factor	rbf	0.0	0.0	0.0	None
degree	int	3	3.0	3.0	3.0	None
gamma	factor	scale	0.0	1.0	1.0	None
coef0	float	0.0	0.0	0.0	0.0	None
shrinking	factor	0	0.0	1.0	1.0	None
probability	factor	0	1.0	1.0	1.0	None
tol	float	0.001	0.0001	0.01	0.00658863194323043	None
cache_size	float	200.0	100.0	400.0	221.52728914167426	None
break_ties	factor	0	0.0	1.0	0.0	None

17.10.1 Show variable importance

```
spot_tuner.plot_importance(threshold=0.025, filename="./figures/" + experiment_name+"_impo
```

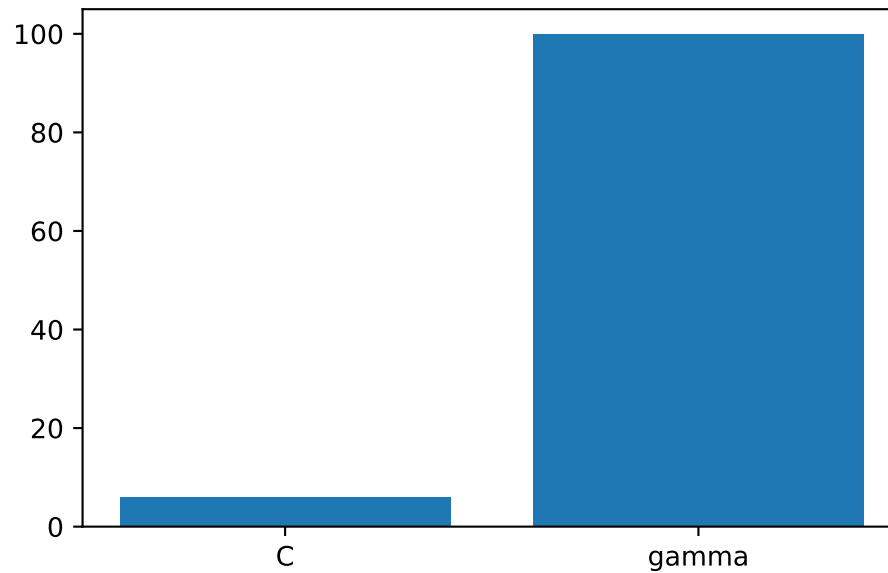



Figure 17.2: Variable importance plot, threshold 0.025.

17.10.2 Get Default Hyperparameters

```
from spotPython.hyperparameters.values import get_default_values, transform_hyper_parameter_values
values_default = get_default_values(fun_control)
values_default = transform_hyper_parameter_values(fun_control=fun_control, hyper_parameters=hyper_parameters)
values_default
```

```
{'C': 1.0,
 'kernel': 'rbf',
 'degree': 3,
 'gamma': 'scale',
 'coef0': 0.0,
 'shrinking': 0,
 'probability': 0,
 'tol': 0.001,
 'cache_size': 200.0,
 'break_ties': 0}
```

```
from sklearn.pipeline import make_pipeline
model_default = make_pipeline(fun_control["prep_model"], fun_control["core_model"](**values_default))
model_default
```

```
Pipeline(steps=[('nonetype', None),
                  ('svc',
                   SVC(break_ties=0, cache_size=200.0, probability=0,
                       shrinking=0))])
```

Note

- Default value for “probability” is False, but we need it to be True for the metric “mapk_score”.

```
values_default.update({"probability": 1})
```

17.10.3 Get SPOT Results

```
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
print(X)
```

```
[[7.64037071e+00 0.00000000e+00 3.00000000e+00 1.00000000e+00
 0.00000000e+00 1.00000000e+00 1.00000000e+00 6.58863194e-03
 2.21527289e+02 0.00000000e+00]]
```

```
from spotPython.hyperparameters.values import assign_values, return_conf_list_from_var_dict
v_dict = assign_values(X, fun_control["var_name"])
return_conf_list_from_var_dict(var_dict=v_dict, fun_control=fun_control)
```

```
[{'C': 7.640370712487234,
  'kernel': 'rbf',
  'degree': 3,
  'gamma': 'auto',
  'coef0': 0.0,
  'shrinking': 1,
  'probability': 1,
  'tol': 0.00658863194323043,
  'cache_size': 221.52728914167426,
  'break_ties': 0}]
```

```

from spotPython.hyperparameters.values import get_one_sklearn_model_from_X
model_spot = get_one_sklearn_model_from_X(X, fun_control)
model_spot

```

```

SVC(C=7.640370712487234, break_ties=0, cache_size=221.52728914167426,
    gamma='auto', probability=1, shrinking=1, tol=0.00658863194323043)

```

17.10.4 Evaluate SPOT Results

- Fetch the data.

```

from spotPython.utils.convert import get_Xy_from_df
X_train, y_train = get_Xy_from_df(fun_control["train"], fun_control["target_column"])
X_test, y_test = get_Xy_from_df(fun_control["test"], fun_control["target_column"])
X_test.shape, y_test.shape

```

```
((177, 64), (177,))
```

- Fit the model with the tuned hyperparameters. This gives one result:

```

model_spot.fit(X_train, y_train)
y_pred = model_spot.predict_proba(X_test)
res = mapk_score(y_true=y_test, y_pred=y_pred, k=3)
res

```

```
0.3578154425612053
```

```

def repeated_eval(n, model):
    res_values = []
    for i in range(n):
        model.fit(X_train, y_train)
        y_pred = model.predict_proba(X_test)
        res = mapk_score(y_true=y_test, y_pred=y_pred, k=3)
        res_values.append(res)
    mean_res = np.mean(res_values)
    print(f"mean_res: {mean_res}")
    std_res = np.std(res_values)
    print(f"std_res: {std_res}")
    min_res = np.min(res_values)

```

```

print(f"min_res: {min_res}")
max_res = np.max(res_values)
print(f"max_res: {max_res}")
median_res = np.median(res_values)
print(f"median_res: {median_res}")
return mean_res, std_res, min_res, max_res, median_res

```

17.10.5 Handling Non-deterministic Results

- Because the model is non-deterministic, we perform $n = 30$ runs and calculate the mean and standard deviation of the performance metric.

```

_ = repeated_eval(30, model_spot)

```

```

mean_res: 0.36459510357815433
std_res: 0.0042834204919603176
min_res: 0.3559322033898305
max_res: 0.3747645951035781
median_res: 0.3644067796610169

```

17.10.6 Evaluation of the Default Hyperparameters

```

model_default["svc"].probability = True
model_default.fit(X_train, y_train)["svc"]

```

```

SVC(break_ties=0, cache_size=200.0, probability=True, shrinking=0)

```

- One evaluation of the default hyperparameters is performed on the hold-out test set.

```

y_pred = model_default.predict_proba(X_test)
mapk_score(y_true=y_test, y_pred=y_pred, k=3)

```

```

0.38888888888888884

```

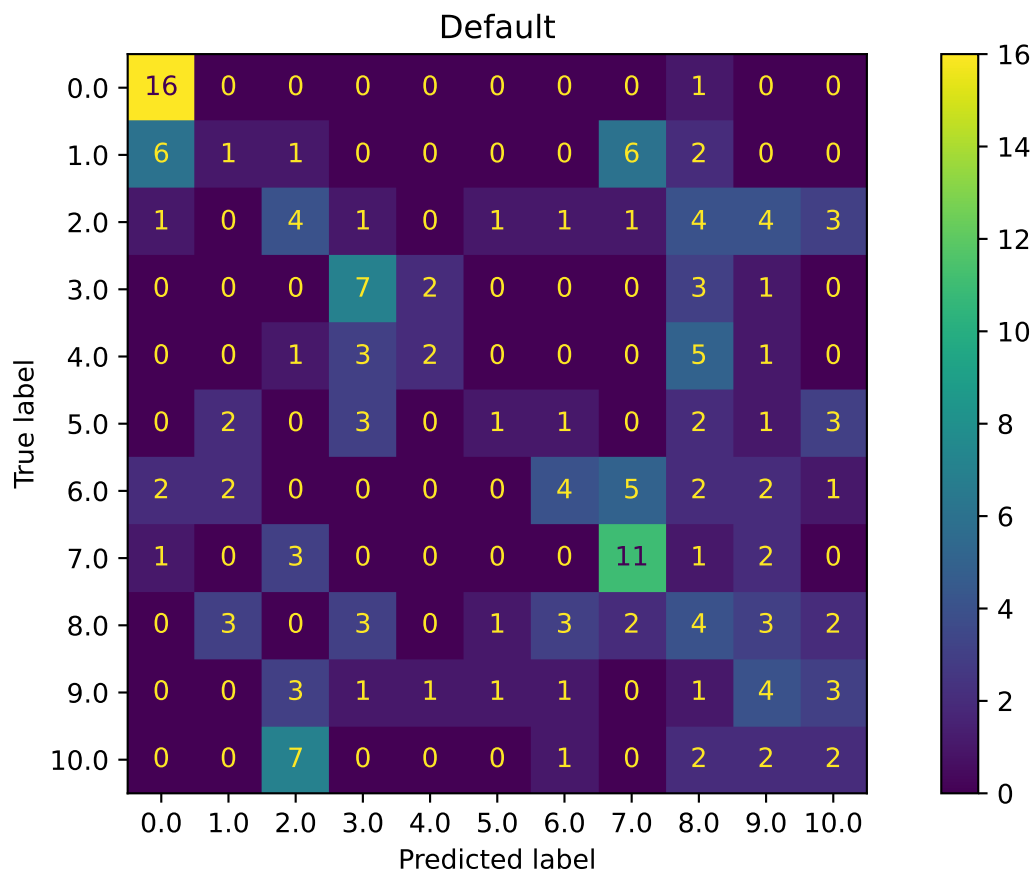
Since one single evaluation is not meaningful, we perform, similar to the evaluation of the SPOT results, $n = 30$ runs of the default setting and calculate the mean and standard deviation of the performance metric.

```
_ = repeated_eval(30, model_default)
```

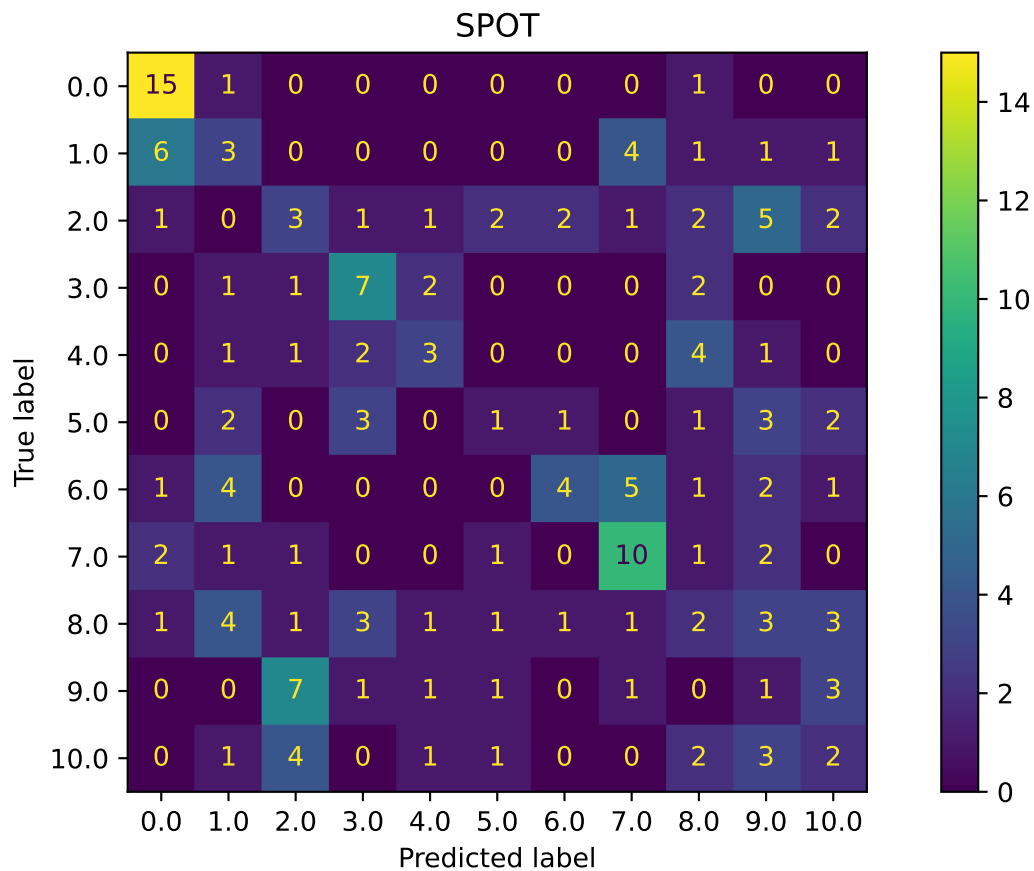
```
mean_res: 0.3848399246704331
std_res: 0.005316330033775537
min_res: 0.37476459510357824
max_res: 0.396421845574388
median_res: 0.3851224105461394
```

17.10.7 Plot: Compare Predictions

```
from spotPython.plot.validation import plot_confusion_matrix
plot_confusion_matrix(model_default, fun_control, title = "Default")
```



```
plot_confusion_matrix(model_spot, fun_control, title="SPOT")
```



```
min(spot_tuner.y), max(spot_tuner.y)
```

```
(-0.38596491228070173, -0.3308270676691729)
```

17.10.8 Cross-validated Evaluations

```
from spotPython.sklearn.traintest import evaluate_cv
fun_control.update({
    "eval": "train_cv",
    "k_folds": 10,
})
```

```
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)
```

(0.3380503144654088, None)

```
fun_control.update({
    "eval": "test_cv",
    "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)
```

(0.35827886710239654, None)

- This is the evaluation that will be used in the comparison:

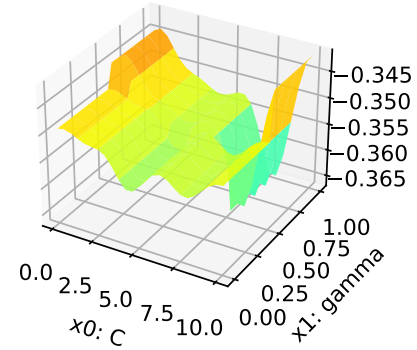
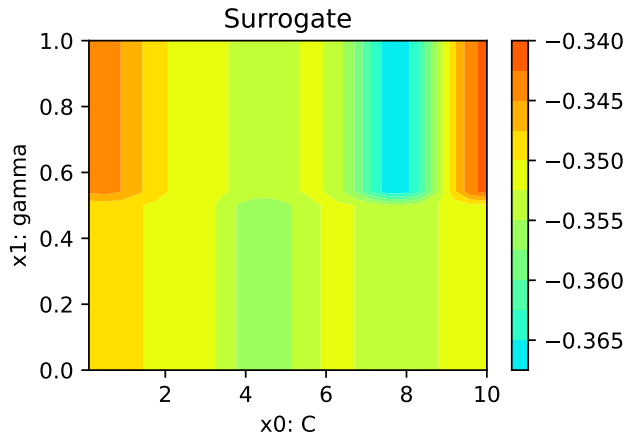
```
fun_control.update({
    "eval": "data_cv",
    "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)
```

(0.3544131455399061, None)

17.10.9 Detailed Hyperparameter Plots

```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

C: 5.992560833072789
gamma: 100.0



17.10.10 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): text/html

17.10.11 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

```
PLOT_ALL = False
if PLOT_ALL:
    n = spot_tuner.k
    for i in range(n-1):
        for j in range(i+1, n):
            spot_tuner.plot_contour(i=i, j=j, min_z=min_z, max_z = max_z)
```


18 HPT: sklearn KNN Classifier VBDP Data

This document refers to the following software versions:

- python: 3.10.10

```
pip list | grep "spot[RiverPython]"
```

spotPython	0.2.52
spotRiver	0.0.94

Note: you may need to restart the kernel to use updated packages.

spotPython can be installed via pip. Alternatively, the source code can be downloaded from gitHub: <https://github.com/sequential-parameter-optimization/spotPython>.

```
!pip install spotPython
```

- Uncomment the following lines if you want to for (re-)installation the latest version of spotPython from gitHub.

```
# import sys
# !{sys.executable} -m pip install --upgrade build
# !{sys.executable} -m pip install --upgrade --force-reinstall spotPython
```

18.1 Step 1: Setup

Before we consider the detailed experimental setup, we select the parameters that affect run time and the initial design size.

```
MAX_TIME = 1
INIT_SIZE = 5
ORIGINAL = False
```

```

import os
import copy
import socket
from datetime import datetime
from dateutil.tz import tzlocal
start_time = datetime.now(tzlocal())
HOSTNAME = socket.gethostname().split(".")[0]
experiment_name = '19-knn-sklearn' + "_" + HOSTNAME + "_" + str(MAX_TIME) + "min_" + str(I
experiment_name = experiment_name.replace(':', '-')
print(experiment_name)
if not os.path.exists('./figures'):
    os.makedirs('./figures')

```

19-knn-sklearn_maans03_1min_5init_2023-07-03_13-05-31

```

import warnings
warnings.filterwarnings("ignore")

```

18.2 Step 2: Initialization of the Empty fun_control Dictionary

 Caution: Tensorboard does not work under Windows

- Since tensorboard does not work under Windows, we recommend setting the parameter `tensorboard_path` to `None` if you are working under Windows.

```

from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="classification",
    tensorboard_path="runs/16_spot_hpt_sklearn_classification")

```

18.2.1 Load Data: Classification VBDP

```

import pandas as pd
if ORIGINAL == True:
    train_df = pd.read_csv('./data/VBDP/trainn.csv')
    test_df = pd.read_csv('./data/VBDP/testt.csv')
else:
    train_df = pd.read_csv('./data/VBDP/train.csv')

```

```

# remove the id column
train_df = train_df.drop(columns=['id'])

from sklearn.preprocessing import OrdinalEncoder
n_samples = train_df.shape[0]
n_features = train_df.shape[1] - 1
target_column = "prognosis"
# Encoder our prognosis labels as integers for easier decoding later
enc = OrdinalEncoder()
train_df[target_column] = enc.fit_transform(train_df[[target_column]])
train_df.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
print(train_df.shape)
train_df.head()

```

(707, 65)

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19
0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0
1	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0
2	0.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0
3	0.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

The full data set `train_df` 64 features. The target column is labeled as `prognosis`.

18.2.2 Holdout Train and Test Data

We split out a hold-out test set (25% of the data) so we can calculate an example MAP@K

```

import numpy as np
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(train_df.drop(target_column, axis=1),
                                                    random_state=42,
                                                    test_size=0.25,
                                                    stratify=train_df[target_column])

train = pd.DataFrame(np.hstack((X_train, np.array(y_train).reshape(-1, 1))))
test = pd.DataFrame(np.hstack((X_test, np.array(y_test).reshape(-1, 1))))
train.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
test.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
print(train.shape)

```

```
print(test.shape)
train.head()
```

```
(530, 65)
```

```
(177, 65)
```

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0
2	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	1.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0
4	0.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0

```
# add the dataset to the fun_control
fun_control.update({"data": train_df, # full dataset,
                  "train": train,
                  "test": test,
                  "n_samples": n_samples,
                  "target_column": target_column})
```

18.3 Step 4: Specification of the Preprocessing Model

Data preprocesssing can be very simple, e.g., you can ignore it. Then you would choose the `prep_model` “None”:

```
prep_model = None
fun_control.update({"prep_model": prep_model})
```

A default approach for numerical data is the `StandardScaler` (mean 0, variance 1). This can be selected as follows:

```
# prep_model = StandardScaler()
# fun_control.update({"prep_model": prep_model})
```

Even more complicated pre-processing steps are possible, e.g., the follwing pipeline:

```
# categorical_columns = []
# one_hot_encoder = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
# prep_model = ColumnTransformer(
```

```
#         transformers=[
#             ("categorical", one_hot_encoder, categorical_columns),
#         ],
#         remainder=StandardScaler(),
#     )
```

18.4 Step 5: Select Model (algorithm) and `core_model_hyper_dict`

The selection of the algorithm (ML model) that should be tuned is done by specifying the its name from the `sklearn` implementation. For example, the SVC support vector machine classifier is selected as follows:

```
fun_control = add_core_model_to_fun_control(SVC, fun_control, SklearnHyperDict)
```

Other `core_models` are, e.g.,:

- `RidgeCV`
- `GradientBoostingRegressor`
- `ElasticNet`
- `RandomForestClassifier`
- `LogisticRegression`
- `KNeighborsClassifier`
- `RandomForestClassifier`
- `GradientBoostingClassifier`
- `HistGradientBoostingClassifier`

We will use the `RandomForestClassifier` classifier in this example.

```
from sklearn.linear_model import RidgeCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.linear_model import ElasticNet
from spotPython.hyperparameters.values import add_core_model_to_fun_control
from spotPython.data.sklearn_hyper_dict import SklearnHyperDict
from spotPython.fun.hypersklearn import HyperSklearn
```

```

# core_model = RidgeCV
# core_model = GradientBoostingRegressor
# core_model = ElasticNet
# core_model = RandomForestClassifier
core_model = KNeighborsClassifier
# core_model = LogisticRegression
# core_model = KNeighborsClassifier
# core_model = GradientBoostingClassifier
# core_model = HistGradientBoostingClassifier
fun_control = add_core_model_to_fun_control(core_model=core_model,
                                           fun_control=fun_control,
                                           hyper_dict=SklearnHyperDict,
                                           filename=None)

```

Now `fun_control` has the information from the JSON file. The available hyperparameters are:

```
print(*fun_control["core_model_hyper_dict"].keys(), sep="\n")
```

```

n_neighbors
weights
algorithm
leaf_size
p

```

18.5 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model`

18.5.1 Modify hyperparameter of type numeric and integer (boolean)

Numeric and boolean values can be modified using the `modify_hyper_parameter_bounds` method. For example, to change the `tol` hyperparameter of the `SVC` model to the interval `[1e-3, 1e-2]`, the following code can be used:

```
fun_control = modify_hyper_parameter_bounds(fun_control, "tol", bounds=[1e-3, 1e-2])
```

```

# from spotPython.hyperparameters.values import modify_hyper_parameter_bounds
# fun_control = modify_hyper_parameter_bounds(fun_control, "probability", bounds=[1, 1])

```

18.5.2 Modify hyperparameter of type factor

`spotPython` provides functions for modifying the hyperparameters, their bounds and factors as well as for activating and de-activating hyperparameters without re-compilation of the Python source code. These functions were described in Section [14.6](#).

Factors can be modified with the `modify_hyper_parameter_levels` function. For example, to exclude the `sigmoid` kernel from the tuning, the `kernel` hyperparameter of the SVC model can be modified as follows:

```
fun_control = modify_hyper_parameter_levels(fun_control, "kernel", ["linear",  
"rbf"])
```

The new setting can be controlled via:

```
fun_control["core_model_hyper_dict"]["kernel"]  
  
# from spotPython.hyperparameters.values import modify_hyper_parameter_levels  
# fun_control = modify_hyper_parameter_levels(fun_control, "kernel", ["rbf"])
```

18.5.3 Optimizers

Optimizers are described in Section [14.6.1](#).

18.5.4 Selection of the Objective: Metric and Loss Functions

- Machine learning models are optimized with respect to a metric, for example, the accuracy function.
- Deep learning, e.g., neural networks are optimized with respect to a loss function, for example, the `cross_entropy` function and evaluated with respect to a metric, for example, the accuracy function.

18.6 Step 7: Selection of the Objective (Loss) Function

The loss function, that is usually used in deep learning for optimizing the weights of the net, is stored in the `fun_control` dictionary as `"loss_function"`.

18.6.1 Metric Function

There are two different types of metrics in `spotPython`:

1. `"metric_river"` is used for the river based evaluation via `eval_oml_iter_progressive`.
2. `"metric_sklearn"` is used for the sklearn based evaluation.

We will consider multi-class classification metrics, e.g., `mapk_score` and `top_k_accuracy_score`.

Predict Probabilities

In this multi-class classification example the machine learning algorithm should return the probabilities of the specific classes (`"predict_proba"`) instead of the predicted values.

We set `"predict_proba"` to `True` in the `fun_control` dictionary.

18.6.1.1 The MAPK Metric

To select the MAPK metric, the following two entries can be added to the `fun_control` dictionary:

```
"metric_sklearn": mapk_score"
```

```
"metric_params": {"k": 3}.
```

18.6.1.2 Other Metrics

Alternatively, other metrics for multi-class classification can be used, e.g., `* top_k_accuracy_score` or `* roc_auc_score`

The metric `roc_auc_score` requires the parameter `"multi_class"`, e.g.,

```
"multi_class": "ovr".
```

This is set in the `fun_control` dictionary.

Weights

`spotPython` performs a minimization, therefore, metrics that should be maximized have to be multiplied by -1. This is done by setting `"weights"` to -1.

- The complete setup for the metric in our example is:


```

from spotPython.utils.metrics import mapk_score
fun_control.update({
    "weights": -1,
    "metric_sklearn": mapk_score,
    "predict_proba": True,
    "metric_params": {"k": 3},
})

```

18.6.2 Evaluation on Hold-out Data

- The default method for computing the performance is "eval_holdout".
- Alternatively, cross-validation can be used for every machine learning model.
- Specifically for RandomForests, the OOB-score can be used.

```

fun_control.update({
    "eval": "train_hold_out",
})

```

18.6.2.1 Cross Validation

Instead of using the OOB-score, the classical cross validation can be used. The number of folds is set by the key "k_folds". For example, to use 5-fold cross validation, the key "k_folds" is set to 5. Uncomment the following line to use cross validation:

```

# fun_control.update({
#     "eval": "train_cv",
#     "k_folds": 10,
# })

```

18.7 Step 8: Calling the SPOT Function

18.7.1 Preparing the SPOT Call

- Get types and variable names as well as lower and upper bounds for the hyperparameters.

```

# extract the variable types, names, and bounds
from spotPython.hyperparameters.values import (get_bound_values,
    get_var_name,

```

```

    get_var_type,)
var_type = get_var_type(fun_control)
var_name = get_var_name(fun_control)
fun_control.update({"var_type": var_type,
                    "var_name": var_name})
lower = get_bound_values(fun_control, "lower")
upper = get_bound_values(fun_control, "upper")

from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control))

```

name	type	default	lower	upper	transform
n_neighbors	int	2	1	7	transform_power_2_int
weights	factor	uniform	0	1	None
algorithm	factor	auto	0	3	None
leaf_size	int	5	2	7	transform_power_2_int
p	int	2	1	2	None

18.7.2 The Objective Function

The objective function is selected next. It implements an interface from `sklearn`'s training, validation, and testing methods to `spotPython`.

```

from spotPython.fun.hyper sklearn import HyperSklearn
fun = HyperSklearn().fun_sklearn

```

18.7.3 Run the Spot Optimizer

- Run SPOT for approx. x mins (`max_time`).
- Note: the run takes longer, because the evaluation time of initial design (here: `init_size`, 20 points) is not considered.

```

from spotPython.hyperparameters.values import get_default_hyperparameters_as_array
hyper_dict=SklearnHyperDict().load()
X_start = get_default_hyperparameters_as_array(fun_control, hyper_dict)
X_start

```

```
array([[2, 0, 0, 5, 2]])
```

```

import numpy as np
from spotPython.spot import spot
from math import inf
spot_tuner = spot.Spot(fun=fun,
                        lower = lower,
                        upper = upper,
                        fun_evals = inf,
                        fun_repeats = 1,
                        max_time = MAX_TIME,
                        noise = False,
                        tolerance_x = np.sqrt(np.spacing(1)),
                        var_type = var_type,
                        var_name = var_name,
                        infill_criterion = "y",
                        n_points = 1,
                        seed=123,
                        log_level = 50,
                        show_models= False,
                        show_progress= True,
                        fun_control = fun_control,
                        design_control={"init_size": INIT_SIZE,
                                      "repeats": 1},
                        surrogate_control={"noise": True,
                                         "cod_type": "norm",
                                         "min_theta": -4,
                                         "max_theta": 3,
                                         "n_theta": len(var_name),
                                         "model_fun_evals": 10_000,
                                         "log_level": 50
                                         })

spot_tuner.run(X_start=X_start)

```

spotPython tuning: -0.3107769423558897 [-----] 0.74%

spotPython tuning: -0.3107769423558897 [-----] 1.59%

spotPython tuning: -0.3107769423558897 [-----] 2.35%

spotPython tuning: -0.3107769423558897 [-----] 3.08%

spotPython tuning: -0.3107769423558897 [-----] 3.83%

spotPython tuning: -0.3107769423558897 [-----] 4.71%

spotPython tuning: -0.3107769423558897 [#-----] 5.93%

spotPython tuning: -0.3107769423558897 [#-----] 6.98%

spotPython tuning: -0.3107769423558897 [#-----] 7.92%

spotPython tuning: -0.3107769423558897 [#-----] 8.81%

spotPython tuning: -0.3107769423558897 [#-----] 9.75%

spotPython tuning: -0.3107769423558897 [#-----] 11.08%

spotPython tuning: -0.3107769423558897 [#-----] 12.48%

spotPython tuning: -0.3107769423558897 [#-----] 14.11%

spotPython tuning: -0.3107769423558897 [##-----] 15.61%

spotPython tuning: -0.3107769423558897 [##-----] 17.22%

spotPython tuning: -0.3107769423558897 [##-----] 19.32%

spotPython tuning: -0.3107769423558897 [##-----] 20.80%

spotPython tuning: -0.3107769423558897 [##-----] 22.61%

spotPython tuning: -0.3107769423558897 [##-----] 24.05%

spotPython tuning: -0.3107769423558897 [###-----] 25.45%

spotPython tuning: -0.3107769423558897 [###-----] 26.65%

spotPython tuning: -0.3107769423558897 [###-----] 27.98%

spotPython tuning: -0.3107769423558897 [###-----] 29.91%

spotPython tuning: -0.3107769423558897 [###-----] 31.36%

spotPython tuning: -0.3107769423558897 [###-----] 33.27%

spotPython tuning: -0.3107769423558897 [###-----] 34.89%

spotPython tuning: -0.3107769423558897 [####-----] 36.75%

spotPython tuning: -0.3107769423558897 [####-----] 38.67%

spotPython tuning: -0.3107769423558897 [####-----] 40.44%

spotPython tuning: -0.3107769423558897 [####-----] 42.08%

spotPython tuning: -0.3107769423558897 [####-----] 44.05%

spotPython tuning: -0.3107769423558897 [#####-----] 45.71%

spotPython tuning: -0.3107769423558897 [#####-----] 47.91%

spotPython tuning: -0.3107769423558897 [#####-----] 50.31%

spotPython tuning: -0.3107769423558897 [#####-----] 52.40%

spotPython tuning: -0.3107769423558897 [#####-----] 54.56%

spotPython tuning: -0.3107769423558897 [#####-----] 56.91%

spotPython tuning: -0.3107769423558897 [#####-----] 59.20%

spotPython tuning: -0.3107769423558897 [#####-----] 61.27%

spotPython tuning: -0.3107769423558897 [#####-----] 63.60%

spotPython tuning: -0.3107769423558897 [#####-----] 65.85%

spotPython tuning: -0.3107769423558897 [#####-----] 67.99%

```

spotPython tuning: -0.3107769423558897 [#####---] 70.33%

spotPython tuning: -0.3107769423558897 [#####---] 72.98%

spotPython tuning: -0.3107769423558897 [#####--] 75.99%

spotPython tuning: -0.3107769423558897 [#####--] 79.25%

spotPython tuning: -0.3107769423558897 [#####--] 82.11%

spotPython tuning: -0.3107769423558897 [#####--] 84.62%

spotPython tuning: -0.3107769423558897 [#####-] 88.07%

spotPython tuning: -0.3107769423558897 [#####] 100.00% Done...

<spotPython.spot.spot.Spot at 0x189129810>

```

18.8 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section 14.9, see also the description in the documentation: [Tensorboard](#).

18.9 Step 10: Results

After the hyperparameter tuning run is finished, the progress of the hyperparameter tuning can be visualized. The following code generates the progress plot from `?@fig-progress`.

```

spot_tuner.plot_progress(log_y=False,
    filename="./figures/" + experiment_name+"_progress.png")

```

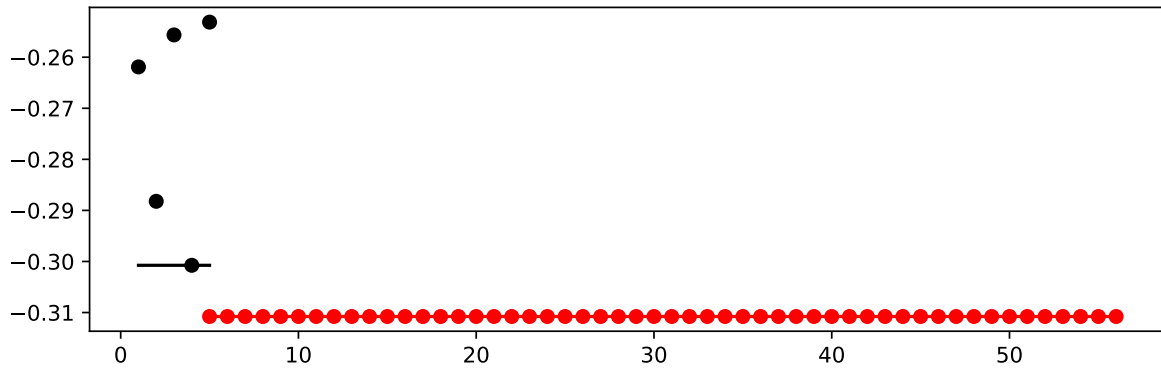


Figure 18.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- Print the results

```
print(gen_design_table(fun_control=fun_control,
                      spot=spot_tuner))
```

name	type	default	lower	upper	tuned	transform
n_neighbors	int	2	1	7	4.0	transform_power_2_int
weights	factor	uniform	0	1	1.0	None
algorithm	factor	auto	0	3	2.0	None
leaf_size	int	5	2	7	6.0	transform_power_2_int
p	int	2	1	2	1.0	None

18.9.1 Show variable importance

```
spot_tuner.plot_importance(threshold=0.025, filename="./figures/" + experiment_name+"_impo
```

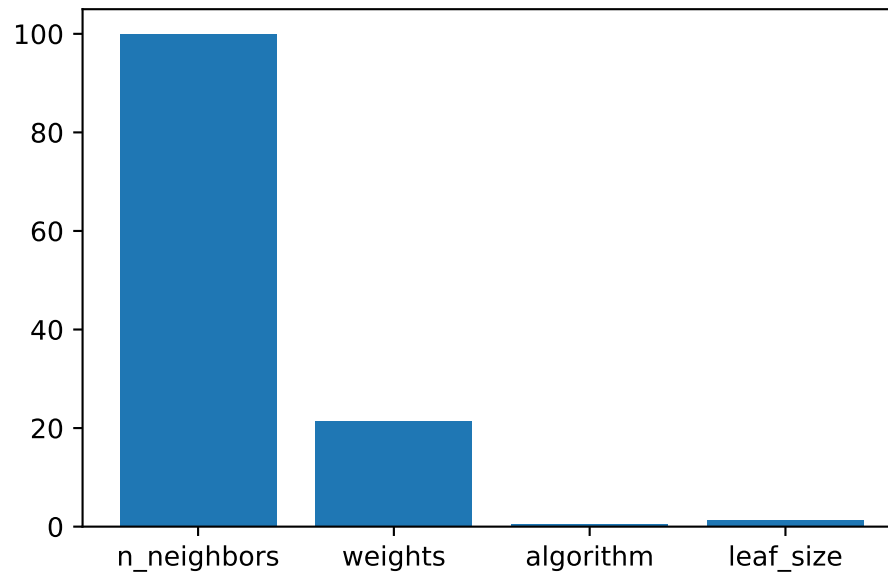


Figure 18.2: Variable importance plot, threshold 0.025.

18.9.2 Get Default Hyperparameters

```
from spotPython.hyperparameters.values import get_default_values, transform_hyper_parameter_values
values_default = get_default_values(fun_control)
values_default = transform_hyper_parameter_values(fun_control=fun_control, hyper_parameters=values_default)
```

```
{'n_neighbors': 4,
 'weights': 'uniform',
 'algorithm': 'auto',
 'leaf_size': 32,
 'p': 2}
```

```
from sklearn.pipeline import make_pipeline
model_default = make_pipeline(fun_control["prep_model"], fun_control["core_model"](**values_default))
model_default
```

```
Pipeline(steps=[('nonetype', None),
                 ('kneighborsclassifier',
                  KNeighborsClassifier(leaf_size=32, n_neighbors=4))])
```


18.9.3 Get SPOT Results

```
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
print(X)
```

```
[[4. 1. 2. 6. 1.]]
```

```
from spotPython.hyperparameters.values import assign_values, return_conf_list_from_var_dict
v_dict = assign_values(X, fun_control["var_name"])
return_conf_list_from_var_dict(var_dict=v_dict, fun_control=fun_control)
```

```
[{'n_neighbors': 16,
  'weights': 'distance',
  'algorithm': 'kd_tree',
  'leaf_size': 64,
  'p': 1}]
```

```
from spotPython.hyperparameters.values import get_one_sklearn_model_from_X
model_spot = get_one_sklearn_model_from_X(X, fun_control)
model_spot
```

```
KNeighborsClassifier(algorithm='kd_tree', leaf_size=64, n_neighbors=16, p=1,
                     weights='distance')
```

18.9.4 Evaluate SPOT Results

- Fetch the data.

```
from spotPython.utils.convert import get_Xy_from_df
X_train, y_train = get_Xy_from_df(fun_control["train"], fun_control["target_column"])
X_test, y_test = get_Xy_from_df(fun_control["test"], fun_control["target_column"])
X_test.shape, y_test.shape
```

```
((177, 64), (177,))
```

- Fit the model with the tuned hyperparameters. This gives one result:

```

model_spot.fit(X_train, y_train)
y_pred = model_spot.predict_proba(X_test)
res = mapk_score(y_true=y_test, y_pred=y_pred, k=3)
res

```

0.3267419962335216

```

def repeated_eval(n, model):
    res_values = []
    for i in range(n):
        model.fit(X_train, y_train)
        y_pred = model.predict_proba(X_test)
        res = mapk_score(y_true=y_test, y_pred=y_pred, k=3)
        res_values.append(res)
    mean_res = np.mean(res_values)
    print(f"mean_res: {mean_res}")
    std_res = np.std(res_values)
    print(f"std_res: {std_res}")
    min_res = np.min(res_values)
    print(f"min_res: {min_res}")
    max_res = np.max(res_values)
    print(f"max_res: {max_res}")
    median_res = np.median(res_values)
    print(f"median_res: {median_res}")
    return mean_res, std_res, min_res, max_res, median_res

```

18.9.5 Handling Non-deterministic Results

- Because the model is non-deterministic, we perform $n = 30$ runs and calculate the mean and standard deviation of the performance metric.

```

_ = repeated_eval(30, model_spot)

```

```

mean_res: 0.3267419962335218
std_res: 1.6653345369377348e-16
min_res: 0.3267419962335216
max_res: 0.3267419962335216
median_res: 0.3267419962335216

```

18.9.6 Evaluation of the Default Hyperparameters

```
model_default.fit(X_train, y_train)["kneighborsclassifier"]
```

```
KNeighborsClassifier(leaf_size=32, n_neighbors=4)
```

- One evaluation of the default hyperparameters is performed on the hold-out test set.

```
y_pred = model_default.predict_proba(X_test)
mapk_score(y_true=y_test, y_pred=y_pred, k=3)
```

```
0.2768361581920904
```

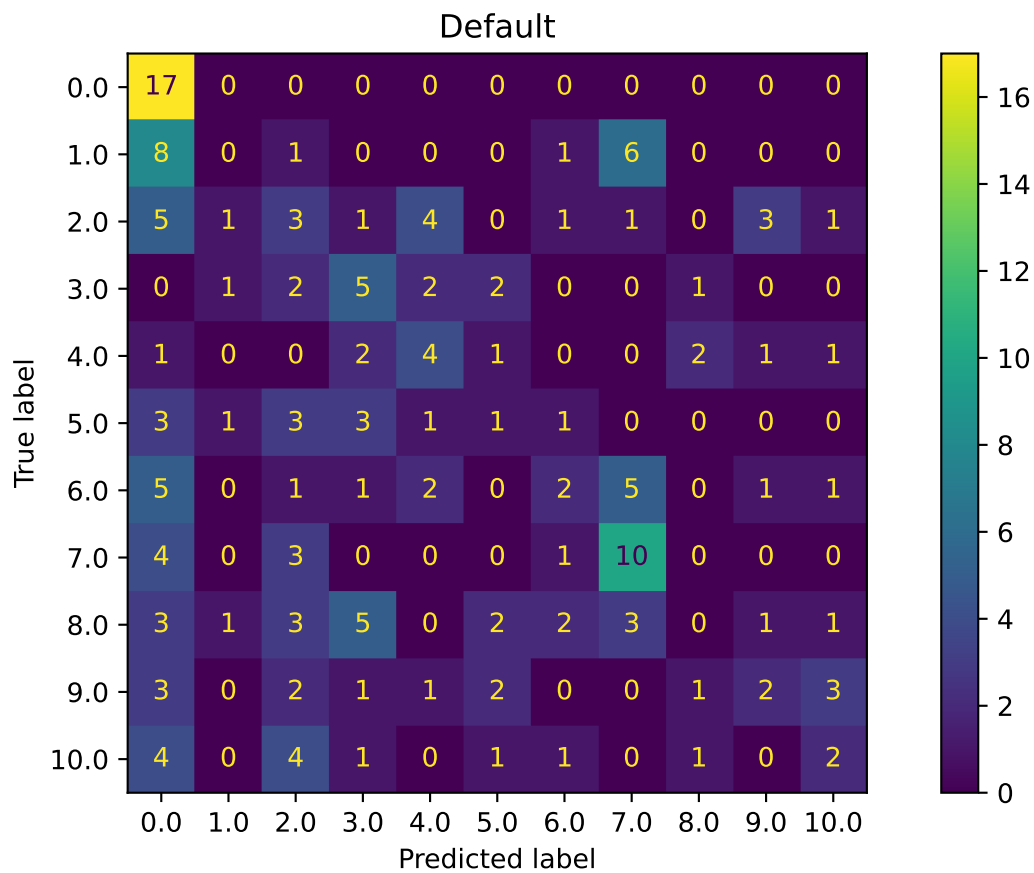
Since one single evaluation is not meaningful, we perform, similar to the evaluation of the SPOT results, $n = 30$ runs of the default setting and calculate the mean and standard deviation of the performance metric.

```
_ = repeated_eval(30, model_default)
```

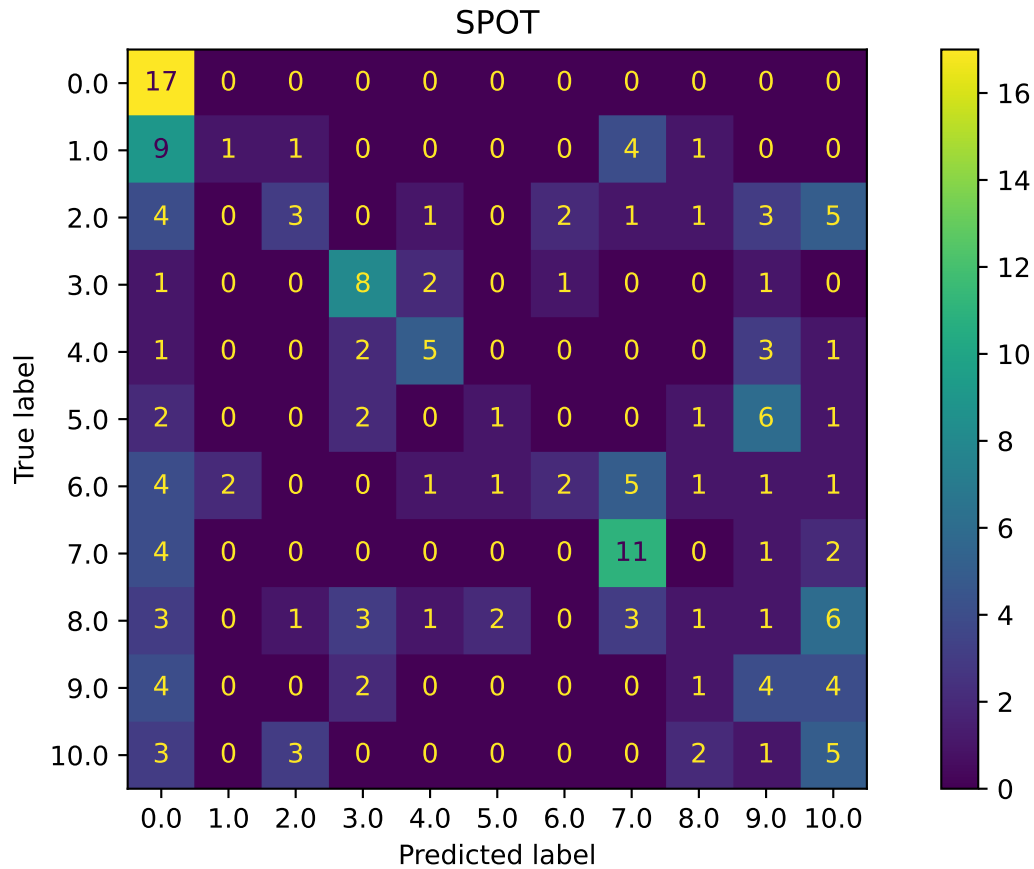
```
mean_res: 0.2768361581920903
std_res: 1.1102230246251565e-16
min_res: 0.2768361581920904
max_res: 0.2768361581920904
median_res: 0.2768361581920904
```

18.9.7 Plot: Compare Predictions

```
from spotPython.plot.validation import plot_confusion_matrix
plot_confusion_matrix(model_default, fun_control, title = "Default")
```



```
plot_confusion_matrix(model_spot, fun_control, title="SPOT")
```



```
min(spot_tuner.y), max(spot_tuner.y)
```

```
(-0.3107769423558897, -0.23558897243107768)
```

18.9.8 Cross-validated Evaluations

```
from spotPython.sklearn.traintest import evaluate_cv
fun_control.update({
    "eval": "train_cv",
    "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)
```

```
(0.3157232704402516, None)
```

```

fun_control.update({
    "eval": "test_cv",
    "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)

```

(0.2832788671023965, None)

- This is the evaluation that will be used in the comparison:

```

fun_control.update({
    "eval": "data_cv",
    "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)

```

(0.3061904761904762, None)

18.9.9 Detailed Hyperparameter Plots

```

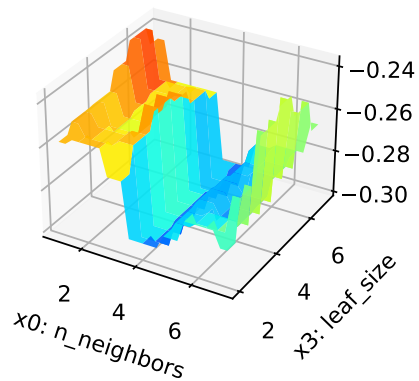
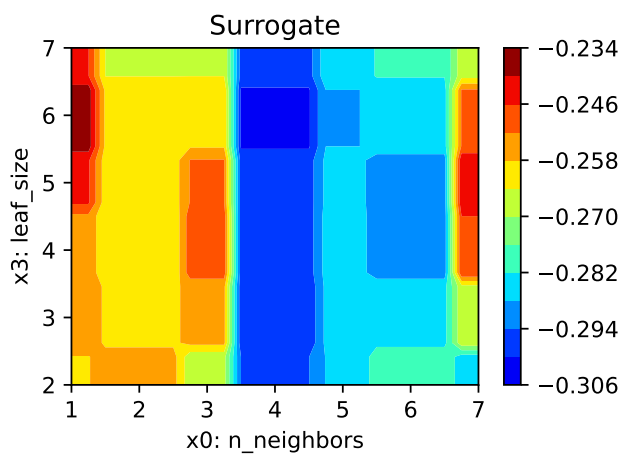
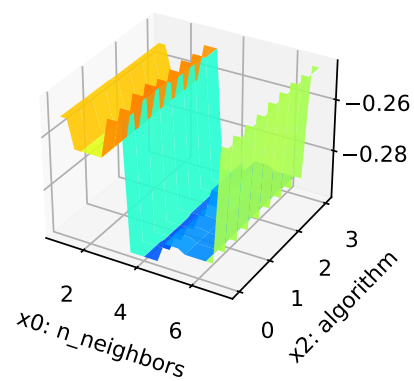
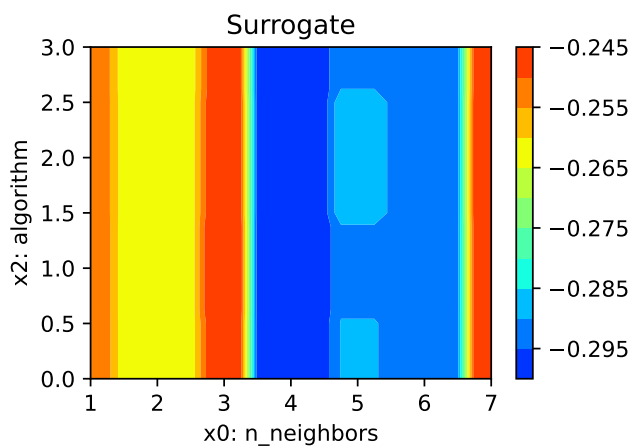
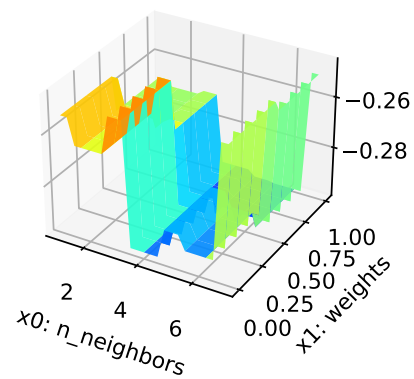
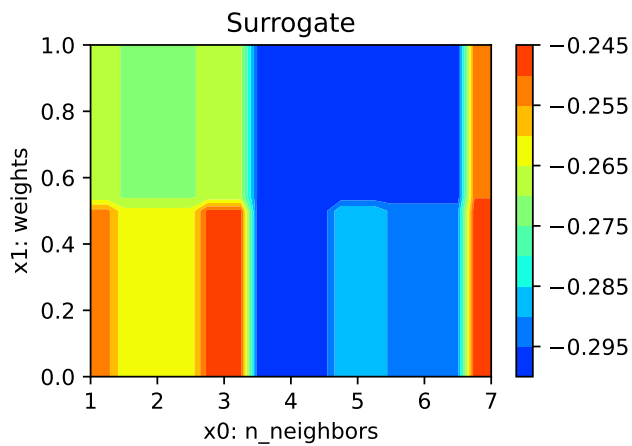
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)

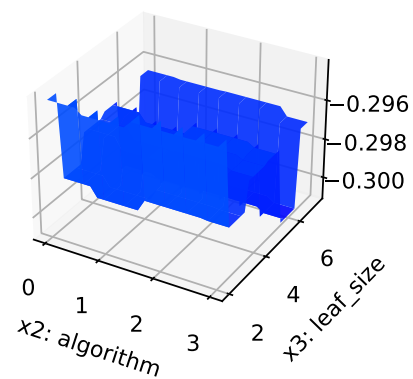
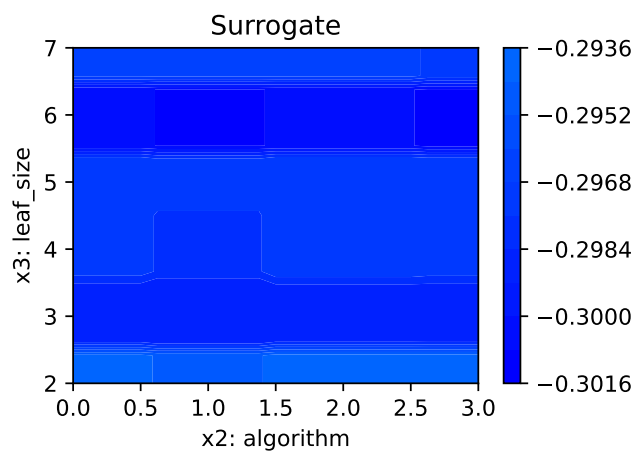
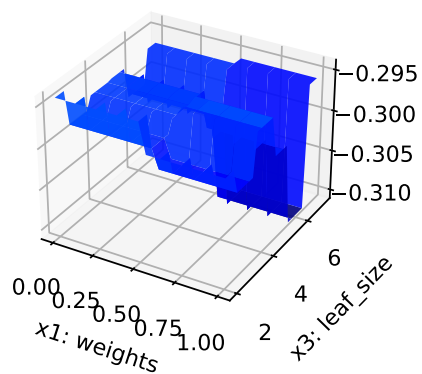
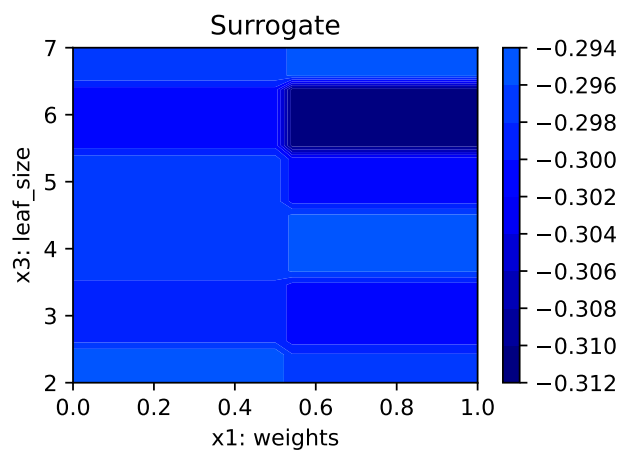
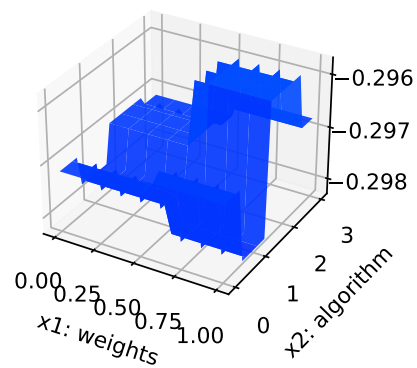
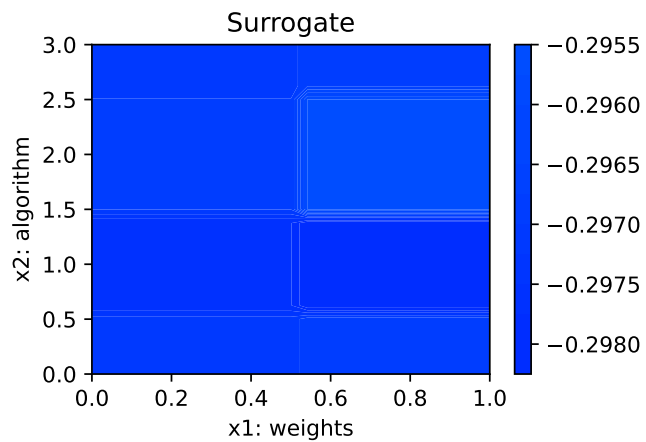
```

```

n_neighbors: 100.0
weights: 21.393178297788836
algorithm: 0.5944758231848488
leaf_size: 1.3747235658929349

```





18.9.10 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): text/html

18.9.11 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

```
PLOT_ALL = False
if PLOT_ALL:
    n = spot_tuner.k
    for i in range(n-1):
        for j in range(i+1, n):
            spot_tuner.plot_contour(i=i, j=j, min_z=min_z, max_z = max_z)
```

19 HPT PyTorch: Regression

In this tutorial, we will show how `spotPython` can be integrated into the PyTorch training workflow for regression tasks.

This document refers to the following software versions:

- python: 3.10.10
- torch: 2.0.1
- torchvision: 0.15.0

```
pip list | grep "spot[RiverPython]"
```

spotPython	0.2.52
spotRiver	0.0.94

Note: you may need to restart the kernel to use updated packages.

`spotPython` can be installed via `pip`. Alternatively, the source code can be downloaded from `gitHub`: <https://github.com/sequential-parameter-optimization/spotPython>.

```
!pip install spotPython
```

- Uncomment the following lines if you want to for (re-)installation the latest version of `spotPython` from `gitHub`.

```
# import sys
# !{sys.executable} -m pip install --upgrade build
# !{sys.executable} -m pip install --upgrade --force-reinstall spotPython
```

19.1 Step 1: Setup

Before we consider the detailed experimental setup, we select the parameters that affect run time, initial design size and the device that is used.

🔥 Caution: Run time and initial design size should be increased for real experiments

- MAX_TIME is set to one minute for demonstration purposes. For real experiments, this should be increased to at least 1 hour.
- INIT_SIZE is set to 5 for demonstration purposes. For real experiments, this should be increased to at least 10.

i Note: Device selection

- The device can be selected by setting the variable DEVICE.
- Since we are using a simple neural net, the setting "cpu" is preferred (on Mac).
- If you have a GPU, you can use "cuda:0" instead.
- If DEVICE is set to None, spotPython will automatically select the device.
 - This might result in "mps" on Macs, which is not the best choice for simple neural nets.

```
MAX_TIME = 1
INIT_SIZE = 5
DEVICE = "cpu" # "cuda:0"
```

```
from spotPython.utils.device import getDevice
DEVICE = getDevice(DEVICE)
print(DEVICE)
```

cpu

```
import os
import copy
import socket
from datetime import datetime
from dateutil.tz import tzlocal
start_time = datetime.now(tzlocal())
HOSTNAME = socket.gethostname().split(".")[0]
experiment_name = '24-torch' + "_" + HOSTNAME + "_" + str(MAX_TIME) + "min_" + str(INIT_SIZE)
experiment_name = experiment_name.replace(':', '-')
print(experiment_name)
if not os.path.exists('./figures'):
    os.makedirs('./figures')
```

24-torch_maans03_1min_5init_2023-07-03_13-16-33

19.2 Step 2: Initialization of the fun_control Dictionary

 Caution: Tensorboard does not work under Windows

- Since tensorboard does not work under Windows, we recommend setting the parameter `tensorboard_path` to `None` if you are working under Windows.

spotPython uses a Python dictionary for storing the information required for the hyperparameter tuning process, which was described in [Section 14.2](#).

```
from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="regression",
    tensorboard_path="runs/24_spot_torch_regression",
    device=DEVICE)
```

19.3 Step 3: PyTorch Data Loading

```
# Create dataset
import pandas as pd
import numpy as np
from sklearn import datasets as sklearn_datasets
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
X, y = sklearn_datasets.make_regression(
    n_samples=1000, n_features=10, noise=1, random_state=123)
y = y.reshape(-1, 1)

# Normalize the data
X_scaler = MinMaxScaler()
X_scaled = X_scaler.fit_transform(X)
y_scaler = MinMaxScaler()
y_scaled = y_scaler.fit_transform(y)

# combine the features and target into a single dataframe named train_df
train_df = pd.DataFrame(np.hstack((X_scaled, y_scaled)))

target_column = "y"
n_samples = train_df.shape[0]
n_features = train_df.shape[1] - 1
```

```

train_df.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
X_train, X_test, y_train, y_test = train_test_split(train_df.drop(target_column,
axis=1),
train_df[target_column],
random_state=42,
test_size=0.25)
trainset = pd.DataFrame(np.hstack((X_train, np.array(y_train).reshape(-1, 1))))
testset = pd.DataFrame(np.hstack((X_test, np.array(y_test).reshape(-1, 1))))
trainset.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
testset.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
print(train_df.shape)
print(trainset.shape)
print(testset.shape)

```

(1000, 11)

(750, 11)

(250, 11)

```

import torch
from spotPython.torch.dataframedataset import DataFrameDataset
dtype_x = torch.float32
dtype_y = torch.float32
train_df = DataFrameDataset(train_df, target_column=target_column,
dtype_x=dtype_x, dtype_y=dtype_y)
train = DataFrameDataset(trainset, target_column=target_column,
dtype_x=dtype_x, dtype_y=dtype_y)
test = DataFrameDataset(testset, target_column=target_column,
dtype_x=dtype_x, dtype_y=dtype_y)
n_samples = len(train)

```

- Now we can test the data loading:

```

from spotPython.torch.traintest import create_train_val_data_loaders
trainloader, testloader = create_train_val_data_loaders(train, 2, True, 0)
for i, data in enumerate(trainloader, 0):
    inputs, labels = data
    print(inputs.shape)
    print(labels.shape)
    print(inputs)
    print(labels)
    break

```

```

torch.Size([2, 10])
torch.Size([2])
tensor([[0.3566, 0.2825, 0.4358, 0.5737, 0.3085, 0.4328, 0.4551, 0.5332, 0.3735,
         0.4098],
        [0.4123, 0.5197, 0.6590, 0.6292, 0.3662, 0.2946, 0.5570, 0.5681, 0.4320,
         0.6783]])
tensor([0.1767, 0.4051])

```

- Since this works fine, we can add the data loading to the `fun_control` dictionary:

```

# add the dataset to the fun_control
fun_control.update({"data": train_df, # full dataset,
                  "train": train,
                  "test": test,
                  "n_samples": n_samples,
                  "target_column": target_column,})

```

19.4 Step 4: Specification of the Preprocessing Model

After the training and test data are specified and added to the `fun_control` dictionary, `spotPython` allows the specification of a data preprocessing pipeline, e.g., for the scaling of the data or for the one-hot encoding of categorical variables, see Section 14.4. This feature is not used here, so we do not change the default value (which is `None`).

19.5 Step 5: Select Model (algorithm) and `core_model_hyper_dict`

19.5.1 Implementing a Configurable Neural Network With `spotPython`

`spotPython` includes the `Net_lin_reg` class which is implemented in the file `netregression.py`.

This class inherits from the class `Net_Core` which is implemented in the file `netcore.py`, see Section 14.5.1.

```

from torch import nn
import spotPython.torch.netcore as netcore

class Net_lin_reg(netcore.Net_Core):
    def __init__(

```

```

        self, _L_in, _L_out, l1, dropout_prob, lr_mult,
        batch_size, epochs, k_folds, patience, optimizer,
        sgd_momentum
    ):
        super(Net_lin_reg, self).__init__(
            lr_mult=lr_mult,
            batch_size=batch_size,
            epochs=epochs,
            k_folds=k_folds,
            patience=patience,
            optimizer=optimizer,
            sgd_momentum=sgd_momentum,
        )
        l2 = max(l1 // 2, 4)
        self.fc1 = nn.Linear(_L_in, l1)
        self.fc2 = nn.Linear(l1, l2)
        self.fc3 = nn.Linear(l2, _L_out)
        self.relu = nn.ReLU()
        self.softmax = nn.Softmax(dim=1)
        self.dropout1 = nn.Dropout(p=dropout_prob)
        self.dropout2 = nn.Dropout(p=dropout_prob / 2)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.dropout1(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.dropout2(x)
        x = self.fc3(x)
        return x

```

19.5.1.1 The Net_Core class

`Net_lin_reg` inherits from the class `Net_Core` which is implemented in the file `netcore.py`. This class was described in Section [14.5.1](#).

```

from spotPython.torch.netregression import Net_lin_reg
from spotPython.data.torch_hyper_dict import TorchHyperDict
from spotPython.hyperparameters.values import add_core_model_to_fun_control
fun_control = add_core_model_to_fun_control(core_model=Net_lin_reg,

```

```
fun_control=fun_control,  
hyper_dict=TorchHyperDict,  
filename=None)
```

19.5.2 The Search Space

19.5.3 Configuring the Search Space With spotPython

19.5.3.1 The hyper_dict Hyperparameters for the Selected Algorithm

spotPython uses JSON files for the specification of the hyperparameters, which were described in Section 14.5.5.

The corresponding entries for the `core_model` class are shown below.

```
fun_control['core_model_hyper_dict']  
  
{ '_L_in': {'type': 'int',  
            'default': 10,  
            'transform': 'None',  
            'lower': 10,  
            'upper': 10},  
  '_L_out': {'type': 'int',  
             'default': 1,  
             'transform': 'None',  
             'lower': 1,  
             'upper': 1},  
  'l1': {'type': 'int',  
         'default': 3,  
         'transform': 'transform_power_2_int',  
         'lower': 3,  
         'upper': 8},  
  'dropout_prob': {'type': 'float',  
                   'default': 0.01,  
                   'transform': 'None',  
                   'lower': 0.0,  
                   'upper': 0.9},  
  'lr_mult': {'type': 'float',  
              'default': 1.0,  
              'transform': 'None',  
              'lower': 0.1,
```



```

    'upper': 10.0},
    'batch_size': {'type': 'int',
                    'default': 4,
                    'transform': 'transform_power_2_int',
                    'lower': 1,
                    'upper': 4},
    'epochs': {'type': 'int',
               'default': 4,
               'transform': 'transform_power_2_int',
               'lower': 4,
               'upper': 9},
    'k_folds': {'type': 'int',
                'default': 1,
                'transform': 'None',
                'lower': 1,
                'upper': 1},
    'patience': {'type': 'int',
                  'default': 2,
                  'transform': 'transform_power_2_int',
                  'lower': 1,
                  'upper': 5},
    'optimizer': {'levels': ['Adadelata',
                              'Adagrad',
                              'Adam',
                              'AdamW',
                              'SparseAdam',
                              'Adamax',
                              'ASGD',
                              'NAdam',
                              'RAdam',
                              'RMSprop',
                              'Rprop',
                              'SGD'],
                  'type': 'factor',
                  'default': 'SGD',
                  'transform': 'None',
                  'class_name': 'torch.optim',
                  'core_model_parameter_type': 'str',
                  'lower': 0,
                  'upper': 12},
    'sgd_momentum': {'type': 'float',
                     'default': 0.0,
                     'transform': 'None',

```

```
'lower': 0.0,  
'upper': 1.0}}
```

19.6 Step 6: Modify hyper_dict Hyperparameters for the Selected Algorithm aka core_model

spotPython provides functions for modifying the hyperparameters, their bounds and factors as well as for activating and de-activating hyperparameters without re-compilation of the Python source code. These functions were described in Section [14.6](#).

19.6.1 Modify hyper_dict Hyperparameters for the Selected Algorithm aka core_model

19.6.1.1 Modify Hyperparameters of Type numeric and integer (boolean)

```
from spotPython.hyperparameters.values import modify_hyper_parameter_bounds  
  
fun_control = modify_hyper_parameter_bounds(fun_control, "epochs", bounds=[2, 16])  
fun_control = modify_hyper_parameter_bounds(fun_control, "patience", bounds=[3, 7])
```

19.6.1.2 Modify Hyperparameter of Type factor

```
from spotPython.hyperparameters.values import modify_hyper_parameter_levels  
fun_control = modify_hyper_parameter_levels(fun_control, "optimizer",  
                                             ["Adadelta", "Adagrad", "Adam", "AdamW", "Adamax", "ASGD", "NAdam"])  
  
fun_control.update({  
    "_L_in": n_features,  
    "_L_out": 1,})
```

19.6.2 Optimizers

Optimizers are described in Section [14.6.1](#).

19.7 Step 7: Selection of the Objective (Loss) Function

19.7.1 Evaluation

The evaluation procedure requires the specification of two elements:

1. the way how the data is split into a train and a test set (see Section [14.7.1](#))
2. the loss function (and a metric).

19.7.2 Loss Functions and Metrics

The key "loss_function" specifies the loss function which is used during the optimization, see Section [14.7.5](#).

We will use MSE loss for the regression task.

```
from torch.nn import MSELoss
loss_torch = MSELoss()
fun_control.update({"loss_function": loss_torch})
```

19.7.3 Metric

```
from torchmetrics import MeanAbsoluteError
metric_torch = MeanAbsoluteError().to(fun_control["device"])
fun_control.update({"metric_torch": metric_torch})
```

19.8 Step 8: Calling the SPOT Function

19.8.1 Preparing the SPOT Call

The following code passes the information about the parameter ranges and bounds to `spot`.

```
# extract the variable types, names, and bounds
from spotPython.hyperparameters.values import (get_bound_values,
        get_var_name,
        get_var_type,)
var_type = get_var_type(fun_control)
var_name = get_var_name(fun_control)
fun_control.update({"var_type": var_type,
```

```

        "var_name": var_name})
lower = get_bound_values(fun_control, "lower")
upper = get_bound_values(fun_control, "upper")

from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control))

```

name	type	default	lower	upper	transform
-----	-----	-----	-----	-----	-----
_L_in	int	10	10	10	None
_L_out	int	1	1	1	None
l1	int	3	3	8	transform_power_2_int
dropout_prob	float	0.01	0	0.9	None
lr_mult	float	1.0	0.1	10	None
batch_size	int	4	1	4	transform_power_2_int
epochs	int	4	2	16	transform_power_2_int
k_folds	int	1	1	1	None
patience	int	2	3	7	transform_power_2_int
optimizer	factor	SGD	0	6	None
sgd_momentum	float	0.0	0	1	None

19.8.2 The Objective Function `fun_torch`

The objective function `fun_torch` is selected next. It implements an interface from PyTorch's training, validation, and testing methods to `spotPython`.

```

from spotPython.fun.hypertorch import HyperTorch
fun = HyperTorch().fun_torch

from spotPython.hyperparameters.values import get_default_hyperparameters_as_array
hyper_dict=TorchHyperDict().load()
X_start = get_default_hyperparameters_as_array(fun_control, hyper_dict)

```

19.8.3 Starting the Hyperparameter Tuning

The `spotPython` hyperparameter tuning is started by calling the `Spot` function as described in Section [14.8.4](#).

```

from spotPython.spot import spot
from math import inf
spot_tuner = spot.Spot(fun=fun,
                        lower = lower,
                        upper = upper,
                        fun_evals = inf,
                        fun_repeats = 1,
                        max_time = MAX_TIME,
                        noise = False,
                        tolerance_x = np.sqrt(np.spacing(1)),
                        var_type = var_type,
                        var_name = var_name,
                        infill_criterion = "y",
                        n_points = 1,
                        seed=123,
                        log_level = 50,
                        show_models= False,
                        show_progress= True,
                        fun_control = fun_control,
                        design_control={"init_size": INIT_SIZE,
                                      "repeats": 1},
                        surrogate_control={"noise": True,
                                         "cod_type": "norm",
                                         "min_theta": -4,
                                         "max_theta": 3,
                                         "n_theta": len(var_name),
                                         "model_fun_evals": 10_000,
                                         "log_level": 50
                                         })

spot_tuner.run(X_start=X_start)

```

```

config: {'_L_in': 10, '_L_out': 1, 'l1': 64, 'dropout_prob': 0.7103122166156, 'lr_mult': 3.6}
Epoch: 1 |

```

```

MeanAbsoluteError: 0.1465502679347992 | Loss: 0.0343003983511344 | Epoch: 2 | MeanAbsoluteError: 0.1465502679347992

```

```

MeanAbsoluteError: 0.1404336094856262 | Loss: 0.0299330577989550 | Epoch: 4 | MeanAbsoluteError: 0.1404336094856262

```

```

MeanAbsoluteError: 0.1188727095723152 | Loss: 0.0235464455388290 | Epoch: 6 | MeanAbsoluteError: 0.1188727095723152

```

MeanAbsoluteError: 0.1182753592729568 | Loss: 0.0222063015068048 | Epoch: 8 | MeanAbsoluteError: 0.1182753592729568 | Loss: 0.0222063015068048 | Epoch: 8 |

MeanAbsoluteError: 0.1292541027069092 | Loss: 0.0255118774152116 | Epoch: 10 | MeanAbsoluteError: 0.1292541027069092 | Loss: 0.0255118774152116 | Epoch: 10 |

MeanAbsoluteError: 0.1036395356059074 | Loss: 0.0182879555137142 | Epoch: 12 | MeanAbsoluteError: 0.1036395356059074 | Loss: 0.0182879555137142 | Epoch: 12 |

MeanAbsoluteError: 0.1133170798420906 | Loss: 0.0217107345877019 | Epoch: 14 | MeanAbsoluteError: 0.1133170798420906 | Loss: 0.0217107345877019 | Epoch: 14 |

MeanAbsoluteError: 0.1110114008188248 | Loss: 0.0201303499174843 | Epoch: 16 | MeanAbsoluteError: 0.1110114008188248 | Loss: 0.0201303499174843 | Epoch: 16 |

MeanAbsoluteError: 0.1097626909613609 | Loss: 0.0190721750994654 | Epoch: 17 | MeanAbsoluteError: 0.1097626909613609 | Loss: 0.0190721750994654 | Epoch: 17 |

MeanAbsoluteError: 0.1022551208734512 | Loss: 0.0169152787604712 | Epoch: 19 | MeanAbsoluteError: 0.1022551208734512 | Loss: 0.0169152787604712 | Epoch: 19 |

MeanAbsoluteError: 0.1047790795564651 | Loss: 0.0172961684875190 | Epoch: 20 | MeanAbsoluteError: 0.1047790795564651 | Loss: 0.0172961684875190 | Epoch: 20 |

MeanAbsoluteError: 0.0924532860517502 | Loss: 0.0140425119806375 | Epoch: 22 | MeanAbsoluteError: 0.0924532860517502 | Loss: 0.0140425119806375 | Epoch: 22 |

Epoch: 23 | MeanAbsoluteError: 0.0938113108277321 | Loss: 0.0150768763116120 | Epoch: 24 | MeanAbsoluteError: 0.0938113108277321 | Loss: 0.0150768763116120 | Epoch: 24 |

MeanAbsoluteError: 0.0874785110354424 | Loss: 0.0137459026484162 | Epoch: 25 | MeanAbsoluteError: 0.0874785110354424 | Loss: 0.0137459026484162 | Epoch: 25 |

Epoch: 26 | MeanAbsoluteError: 0.0949684306979179 | Loss: 0.0159820871256096 | Epoch: 27 | MeanAbsoluteError: 0.0949684306979179 | Loss: 0.0159820871256096 | Epoch: 27 |

MeanAbsoluteError: 0.0920959487557411 | Loss: 0.0142600782718019 | Epoch: 29 | MeanAbsoluteError: 0.0920959487557411 | Loss: 0.0142600782718019 | Epoch: 29 |

MeanAbsoluteError: 0.0888112634420395 | Loss: 0.0130078479230992 | Epoch: 32 | MeanAbsoluteError: 0.0888112634420395 | Loss: 0.0130078479230992 | Epoch: 32 |

Epoch: 35 | MeanAbsoluteError: 0.0841168686747551 | Loss: 0.0125765538909227 | Epoch: 36 | MeanAbsoluteError: 0.0841168686747551 | Loss: 0.0125765538909227 | Epoch: 36 |

Epoch: 38 | MeanAbsoluteError: 0.0824426338076591 | Loss: 0.0121180862647873 | Epoch: 39 | MeanAbsoluteError: 0.0824426338076591 | Loss: 0.0121180862647873 | Epoch: 39 |

Epoch: 41 | MeanAbsoluteError: 0.0707920864224434 | Loss: 0.0094240001635626 | Epoch: 42 | MeanAbsoluteError: 0.0707920864224434 | Loss: 0.0094240001635626 | Epoch: 42 |

MeanAbsoluteError: 0.0698742344975471 | Loss: 0.0089140196367608 | Epoch: 45 | MeanAbsoluteError: 0.0698742344975471 | Loss: 0.0089140196367608 | Epoch: 45 |

MeanAbsoluteError: 0.0736887753009796 | Loss: 0.0098165841742517 | Epoch: 48 | MeanAbsoluteError: 0.0736887753009796 | Loss: 0.0098165841742517 | Epoch: 48 |

Epoch: 51 | MeanAbsoluteError: 0.0710147842764854 | Loss: 0.0090036800033168 | Epoch: 52 | MeanAbsoluteError: 0.0727041661739349 | Loss: 0.0103133383979041 | Epoch: 54 | MeanAbsoluteError: 0.0675030723214149 | Loss: 0.0083705220359231 | Epoch: 58 | MeanAbsoluteError: 0.0714833438396454 | Loss: 0.0100503236432192 | Early stopping at epoch 58
Returned to Spot: Validation loss: 0.010050323643219216

config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.19981931523998656, 'lr_mult': 1.0}

Epoch: 1 | MeanAbsoluteError: 0.2298645228147507 | Loss: 0.0723050262190794 | Epoch: 2 | MeanAbsoluteError: 0.1171868517994881 | Loss: 0.0211990836419557 | Epoch: 7 | MeanAbsoluteError: 0.0957288071513176 | Loss: 0.0143535225407073 | Epoch: 14 | MeanAbsoluteError: 0.0865119770169258 | Loss: 0.0105804685307176 | Epoch: 21 | MeanAbsoluteError: 0.0539892278611660 | Loss: 0.0053302555614592 | Epoch: 28 | MeanAbsoluteError: 0.0551659502089024 | Loss: 0.0057379902818387 | Epoch: 35 | MeanAbsoluteError: 0.0445129461586475 | Loss: 0.0032646789730183 | Epoch: 42 | MeanAbsoluteError: 0.0720015540719032 | Loss: 0.0075748084407104 | Epoch: 49 | MeanAbsoluteError: 0.0848084315657616 | Loss: 0.0096277989888270 | Epoch: 55 | MeanAbsoluteError: 0.0751864090561867 | Loss: 0.0072491896142693 | Epoch: 62 | MeanAbsoluteError: 0.0653484761714935 | Loss: 0.0059973378175575 | Epoch: 68 | MeanAbsoluteError: 0.0623837485909462 | Loss: 0.0060014659171238 | Epoch: 75 | MeanAbsoluteError: 0.0447830818593502 | Loss: 0.0032139029476399 | Epoch: 82 | MeanAbsoluteError: 0.0615296885371208 | Loss: 0.0067420810902197 | Epoch: 89 | MeanAbsoluteError:

```

MeanAbsoluteError: 0.0698826834559441 | Loss: 0.0068900385062749 | Epoch: 96 | MeanAbsoluteE
MeanAbsoluteError: 0.0643774047493935 | Loss: 0.0066801846444019 | Epoch: 103 | MeanAbsoluteE
MeanAbsoluteError: 0.0372107438743114 | Loss: 0.0022350637959071 | Epoch: 110 | MeanAbsoluteE
MeanAbsoluteError: 0.0289837904274464 | Loss: 0.0017579200074052 | Epoch: 117 | MeanAbsoluteE
MeanAbsoluteError: 0.0820707380771637 | Loss: 0.0088259873323535 | Epoch: 124 | MeanAbsoluteE
MeanAbsoluteError: 0.0765396505594254 | Loss: 0.0102964709512889 | Epoch: 130 | MeanAbsoluteE
MeanAbsoluteError: 0.0627372190356255 | Loss: 0.0058779554613131 | Epoch: 137 | MeanAbsoluteE
MeanAbsoluteError: 0.0292934514582157 | Loss: 0.0015782496852090 | Epoch: 144 | MeanAbsoluteE
MeanAbsoluteError: 0.0416226498782635 | Loss: 0.0027181418232718 | Epoch: 151 | MeanAbsoluteE
MeanAbsoluteError: 0.0530892871320248 | Loss: 0.0049227942516537 | Epoch: 158 | MeanAbsoluteE
MeanAbsoluteError: 0.0392248518764973 | Loss: 0.0029431987235225 | Epoch: 165 | MeanAbsoluteE
MeanAbsoluteError: 0.0465169586241245 | Loss: 0.0035185340522347 | Epoch: 172 | MeanAbsoluteE
MeanAbsoluteError: 0.0306412409991026 | Loss: 0.0016865455361671 | Early stopping at epoch 17
Returned to Spot: Validation loss: 0.0016865455361671354

config: {'_L_in': 10, '_L_out': 1, 'l1': 128, 'dropout_prob': 0.8582565260508446, 'lr_mult':
Epoch: 1 |

MeanAbsoluteError: 0.2102675139904022 | Loss: 0.0700052329490427 | Epoch: 2 |

MeanAbsoluteError: 0.1902111917734146 | Loss: 0.0578029420127859 | Epoch: 3 |

MeanAbsoluteError: 0.1704472601413727 | Loss: 0.0455920836143196 | Epoch: 4 |

MeanAbsoluteError: 0.1693751066923141 | Loss: 0.0453828522380597 | Epoch: 5 |

```


MeanAbsoluteError: 0.1519162058830261 | Loss: 0.0370325942806570 | Epoch: 6 |

MeanAbsoluteError: 0.1647857576608658 | Loss: 0.0413459200109355 | Epoch: 7 |

MeanAbsoluteError: 0.1444289833307266 | Loss: 0.0329250520459997 | Epoch: 8 |

MeanAbsoluteError: 0.1495256125926971 | Loss: 0.0346060129922019 | Epoch: 9 |

MeanAbsoluteError: 0.1445829421281815 | Loss: 0.0327894075153745 | Epoch: 10 |

MeanAbsoluteError: 0.1404394507408142 | Loss: 0.0309249591022247 | Epoch: 11 |

MeanAbsoluteError: 0.1367861032485962 | Loss: 0.0301508035687342 | Epoch: 12 |

MeanAbsoluteError: 0.1403637528419495 | Loss: 0.0298146008420736 | Epoch: 13 |

MeanAbsoluteError: 0.1421889811754227 | Loss: 0.0309560968841348 | Epoch: 14 |

MeanAbsoluteError: 0.1355681270360947 | Loss: 0.0287911524920492 | Epoch: 15 |

MeanAbsoluteError: 0.1322758793830872 | Loss: 0.0269071043825049 | Epoch: 16 |

MeanAbsoluteError: 0.1372147947549820 | Loss: 0.0299760351347504 | Epoch: 17 |

MeanAbsoluteError: 0.1334013938903809 | Loss: 0.0284501889260719 | Epoch: 18 |

MeanAbsoluteError: 0.1335695087909698 | Loss: 0.0275006578292232 | Epoch: 19 |

MeanAbsoluteError: 0.1363352984189987 | Loss: 0.0290654015521189 | Epoch: 20 |

MeanAbsoluteError: 0.1326225101947784 | Loss: 0.0275961426610108 | Epoch: 21 |

MeanAbsoluteError: 0.1369360089302063 | Loss: 0.0287218223907015 | Epoch: 22 |

MeanAbsoluteError: 0.1323734968900681 | Loss: 0.0278122280626364 | Epoch: 23 |

MeanAbsoluteError: 0.1373230367898941 | Loss: 0.0291733847224774 | Epoch: 24 |

MeanAbsoluteError: 0.1348941922187805 | Loss: 0.0287413676993068 | Epoch: 25 |

MeanAbsoluteError: 0.1398703008890152 | Loss: 0.0300548223855731 | Epoch: 26 |

MeanAbsoluteError: 0.1359839737415314 | Loss: 0.0278233862900136 | Epoch: 27 |

MeanAbsoluteError: 0.1371723413467407 | Loss: 0.0295950265634262 | Epoch: 28 |

MeanAbsoluteError: 0.1320696026086807 | Loss: 0.0275795390935915 | Epoch: 29 |

MeanAbsoluteError: 0.1351308971643448 | Loss: 0.0282096868621496 | Epoch: 30 |

MeanAbsoluteError: 0.1336297243833542 | Loss: 0.0284704427943022 | Epoch: 31 |

MeanAbsoluteError: 0.1294516921043396 | Loss: 0.0259481846124011 | Epoch: 32 |

MeanAbsoluteError: 0.1316147893667221 | Loss: 0.0278283554319448 | Epoch: 33 |

MeanAbsoluteError: 0.1346359699964523 | Loss: 0.0281896035256796 | Epoch: 34 |

MeanAbsoluteError: 0.1319144815206528 | Loss: 0.0273319061663157 | Epoch: 35 |

MeanAbsoluteError: 0.1326851397752762 | Loss: 0.0276891666042502 | Epoch: 36 |

MeanAbsoluteError: 0.1345913410186768 | Loss: 0.0280713521813353 | Epoch: 37 |

MeanAbsoluteError: 0.1326993554830551 | Loss: 0.0272130460611243 | Epoch: 38 |

MeanAbsoluteError: 0.1354285031557083 | Loss: 0.0286762683039221 | Epoch: 39 |

MeanAbsoluteError: 0.1343396306037903 | Loss: 0.0292079061026501 | Epoch: 40 |

MeanAbsoluteError: 0.1323216855525970 | Loss: 0.0271999799676511 | Epoch: 41 |

MeanAbsoluteError: 0.1359363347291946 | Loss: 0.0287333529506577 | Epoch: 42 |

MeanAbsoluteError: 0.1383922100067139 | Loss: 0.0297411425444686 | Epoch: 43 |

MeanAbsoluteError: 0.1332575231790543 | Loss: 0.0271407835496939 | Epoch: 44 |

MeanAbsoluteError: 0.1348319053649902 | Loss: 0.0285623567773534 | Epoch: 45 |

MeanAbsoluteError: 0.1325636655092239 | Loss: 0.0269271087023662 | Epoch: 46 |

MeanAbsoluteError: 0.1323404908180237 | Loss: 0.0273918965288127 | Epoch: 47 |

MeanAbsoluteError: 0.1314835548400879 | Loss: 0.0275082659434217 | Epoch: 48 |

MeanAbsoluteError: 0.1325118839740753 | Loss: 0.0277585854500649 | Epoch: 49 |

MeanAbsoluteError: 0.1351818889379501 | Loss: 0.0288086264085723 | Epoch: 50 |

MeanAbsoluteError: 0.1328666657209396 | Loss: 0.0270632607707133 | Epoch: 51 |

MeanAbsoluteError: 0.1315873265266418 | Loss: 0.0271051560469399 | Epoch: 52 |

MeanAbsoluteError: 0.1309643387794495 | Loss: 0.0270515738606142 | Epoch: 53 |

MeanAbsoluteError: 0.1308590173721313 | Loss: 0.0267266542107488 | Epoch: 54 |

MeanAbsoluteError: 0.1291332244873047 | Loss: 0.0262955247475960 | Epoch: 55 |

MeanAbsoluteError: 0.1332070380449295 | Loss: 0.0278715249624414 | Epoch: 56 |

MeanAbsoluteError: 0.1315446794033051 | Loss: 0.0265039573529793 | Epoch: 57 |

MeanAbsoluteError: 0.1346282660961151 | Loss: 0.0280348556671136 | Epoch: 58 |

MeanAbsoluteError: 0.1319085508584976 | Loss: 0.0279327961548794 | Epoch: 59 |

MeanAbsoluteError: 0.1346115320920944 | Loss: 0.0284140881807131 | Epoch: 60 |

MeanAbsoluteError: 0.1329792886972427 | Loss: 0.0274519832962081 | Epoch: 61 |

MeanAbsoluteError: 0.1313138455152512 | Loss: 0.0273683139205483 | Epoch: 62 |

MeanAbsoluteError: 0.1324835121631622 | Loss: 0.0272303155153349 | Epoch: 63 |

MeanAbsoluteError: 0.1308646500110626 | Loss: 0.0263581244484521 | Epoch: 64 |

MeanAbsoluteError: 0.1340015381574631 | Loss: 0.0274037616678591 | Epoch: 65 |

MeanAbsoluteError: 0.1322430223226547 | Loss: 0.0272585186181823 | Epoch: 66 |

MeanAbsoluteError: 0.1358350366353989 | Loss: 0.0285829973557459 | Epoch: 67 |

MeanAbsoluteError: 0.1313312202692032 | Loss: 0.0262801016263256 | Epoch: 68 |

MeanAbsoluteError: 0.1326214373111725 | Loss: 0.0282333546235653 | Epoch: 69 |

MeanAbsoluteError: 0.1294749230146408 | Loss: 0.0274280032874473 | Epoch: 70 |

MeanAbsoluteError: 0.1334284991025925 | Loss: 0.0276461564169343 | Epoch: 71 |

MeanAbsoluteError: 0.1350211650133133 | Loss: 0.0280860106343607 | Epoch: 72 |

MeanAbsoluteError: 0.1288627386093140 | Loss: 0.0262988397310073 | Epoch: 73 |

MeanAbsoluteError: 0.1258650571107864 | Loss: 0.0257986366617843 | Epoch: 74 |

MeanAbsoluteError: 0.1320160180330276 | Loss: 0.0284305243047735 | Epoch: 75 |

MeanAbsoluteError: 0.1315242499113083 | Loss: 0.0267569974019716 | Epoch: 76 |

MeanAbsoluteError: 0.1270900815725327 | Loss: 0.0251626516123846 | Epoch: 77 |

MeanAbsoluteError: 0.1290774196386337 | Loss: 0.0262120977829909 | Epoch: 78 |

MeanAbsoluteError: 0.1323226839303970 | Loss: 0.0267147681474065 | Epoch: 79 |

MeanAbsoluteError: 0.1293153911828995 | Loss: 0.0263596312516165 | Epoch: 80 |

MeanAbsoluteError: 0.1291421055793762 | Loss: 0.0266981449957530 | Epoch: 81 |

MeanAbsoluteError: 0.1317014396190643 | Loss: 0.0267117979713172 | Epoch: 82 |

MeanAbsoluteError: 0.1329379826784134 | Loss: 0.0267283582722303 | Epoch: 83 |

MeanAbsoluteError: 0.1293889582157135 | Loss: 0.0268393121918659 | Epoch: 84 |

MeanAbsoluteError: 0.1289502680301666 | Loss: 0.0265247344132513 | Epoch: 85 |

MeanAbsoluteError: 0.1309358477592468 | Loss: 0.0267960062957718 | Epoch: 86 |

MeanAbsoluteError: 0.1339800208806992 | Loss: 0.0291091277476638 | Epoch: 87 |

MeanAbsoluteError: 0.1306488364934921 | Loss: 0.0276701888881507 | Epoch: 88 |

MeanAbsoluteError: 0.1269007325172424 | Loss: 0.0251975345493580 | Epoch: 89 |

MeanAbsoluteError: 0.1317367702722549 | Loss: 0.0266242250544140 | Epoch: 90 |

MeanAbsoluteError: 0.1306271106004715 | Loss: 0.0258505436492851 | Epoch: 91 |

MeanAbsoluteError: 0.1257923096418381 | Loss: 0.0254803450923525 | Epoch: 92 |

MeanAbsoluteError: 0.1319987326860428 | Loss: 0.0263663655910447 | Epoch: 93 |

MeanAbsoluteError: 0.1297155171632767 | Loss: 0.0262844181972711 | Epoch: 94 |

MeanAbsoluteError: 0.1282313317060471 | Loss: 0.0258336456426575 | Epoch: 95 |

MeanAbsoluteError: 0.1305791288614273 | Loss: 0.0259409813903039 | Epoch: 96 |

MeanAbsoluteError: 0.1299192458391190 | Loss: 0.0267952538813309 | Epoch: 97 |

MeanAbsoluteError: 0.1337317526340485 | Loss: 0.0285358113419109 | Epoch: 98 |

MeanAbsoluteError: 0.1359720528125763 | Loss: 0.0284682803261482 | Epoch: 99 |

MeanAbsoluteError: 0.1368359178304672 | Loss: 0.0274737656553043 | Epoch: 100 |

MeanAbsoluteError: 0.1246303915977478 | Loss: 0.0250178357934177 | Epoch: 101 |

MeanAbsoluteError: 0.1314410716295242 | Loss: 0.0273502034942309 | Epoch: 102 |

MeanAbsoluteError: 0.1310835480690002 | Loss: 0.0265947020305612 | Epoch: 103 |

MeanAbsoluteError: 0.1256079375743866 | Loss: 0.0248133385669886 | Epoch: 104 |

MeanAbsoluteError: 0.1273702085018158 | Loss: 0.0258738804601793 | Epoch: 105 |

MeanAbsoluteError: 0.1329370141029358 | Loss: 0.0266674335473605 | Epoch: 106 |

MeanAbsoluteError: 0.1300746947526932 | Loss: 0.0260796583358509 | Epoch: 107 |

MeanAbsoluteError: 0.1262635737657547 | Loss: 0.0252232829822848 | Epoch: 108 |

MeanAbsoluteError: 0.1316016614437103 | Loss: 0.0267054219685391 | Epoch: 109 |

MeanAbsoluteError: 0.1257272958755493 | Loss: 0.0246715343375884 | Epoch: 110 |

MeanAbsoluteError: 0.1216469556093216 | Loss: 0.0241570896546909 | Epoch: 111 |

MeanAbsoluteError: 0.1300028711557388 | Loss: 0.0257637493615039 | Epoch: 112 |

MeanAbsoluteError: 0.1254722923040390 | Loss: 0.0244095791159392 | Epoch: 113 |

MeanAbsoluteError: 0.1247556433081627 | Loss: 0.0243579811501938 | Epoch: 114 |

MeanAbsoluteError: 0.1298024356365204 | Loss: 0.0257387680254760 | Epoch: 115 |

MeanAbsoluteError: 0.1292264163494110 | Loss: 0.0258599332454226 | Epoch: 116 |

MeanAbsoluteError: 0.1263683587312698 | Loss: 0.0237060754367849 | Epoch: 117 |

MeanAbsoluteError: 0.1295398175716400 | Loss: 0.0254856416733674 | Epoch: 118 |

MeanAbsoluteError: 0.1306554675102234 | Loss: 0.0269362826354336 | Epoch: 119 |

MeanAbsoluteError: 0.1246975362300873 | Loss: 0.0245332025622095 | Epoch: 120 |

MeanAbsoluteError: 0.1276959031820297 | Loss: 0.0257547269590092 | Epoch: 121 |

MeanAbsoluteError: 0.1313905268907547 | Loss: 0.0263389125947530 | Epoch: 122 |

MeanAbsoluteError: 0.1308957487344742 | Loss: 0.0259577891469235 | Epoch: 123 |

MeanAbsoluteError: 0.1270331889390945 | Loss: 0.0255616329296026 | Epoch: 124 |

MeanAbsoluteError: 0.1258517652750015 | Loss: 0.0248064573726151 | Epoch: 125 |

MeanAbsoluteError: 0.1281343549489975 | Loss: 0.0259041373229653 | Epoch: 126 |

MeanAbsoluteError: 0.1216480880975723 | Loss: 0.0236343797259906 | Epoch: 127 |

MeanAbsoluteError: 0.1280666887760162 | Loss: 0.0255383877176791 | Epoch: 128 |

MeanAbsoluteError: 0.1275421082973480 | Loss: 0.0249033994598237 | Epoch: 129 |

MeanAbsoluteError: 0.1286983937025070 | Loss: 0.0250221212751543 | Epoch: 130 |

MeanAbsoluteError: 0.1285388320684433 | Loss: 0.0263967684063634 | Epoch: 131 |

MeanAbsoluteError: 0.1292266100645065 | Loss: 0.0251264087326369 | Epoch: 132 |

MeanAbsoluteError: 0.1295073479413986 | Loss: 0.0253726596887767 | Epoch: 133 |

MeanAbsoluteError: 0.1279838830232620 | Loss: 0.0255606904519664 | Epoch: 134 |

MeanAbsoluteError: 0.1300766915082932 | Loss: 0.0256406949323718 | Epoch: 135 |

MeanAbsoluteError: 0.1244168356060982 | Loss: 0.0241338649195192 | Epoch: 136 |

MeanAbsoluteError: 0.1242953985929489 | Loss: 0.0250053166821211 | Epoch: 137 |

MeanAbsoluteError: 0.1281491816043854 | Loss: 0.0251866012951359 | Epoch: 138 |

MeanAbsoluteError: 0.1280431002378464 | Loss: 0.0251377001110920 | Epoch: 139 |

MeanAbsoluteError: 0.1248274371027946 | Loss: 0.0241240021228441 | Epoch: 140 |

MeanAbsoluteError: 0.1227015554904938 | Loss: 0.0245439655071459 | Epoch: 141 |

MeanAbsoluteError: 0.1298843622207642 | Loss: 0.0255411586355573 | Epoch: 142 |

MeanAbsoluteError: 0.1220896765589714 | Loss: 0.0243916879071427 | Epoch: 143 |

MeanAbsoluteError: 0.1272732764482498 | Loss: 0.0256080518243834 | Epoch: 144 |

MeanAbsoluteError: 0.1272633075714111 | Loss: 0.0250219793947933 | Epoch: 145 |

MeanAbsoluteError: 0.1243763342499733 | Loss: 0.0249034954612337 | Epoch: 146 |

MeanAbsoluteError: 0.1276964098215103 | Loss: 0.0247417030789044 | Epoch: 147 |

MeanAbsoluteError: 0.1216303780674934 | Loss: 0.0235365609275565 | Epoch: 148 |

MeanAbsoluteError: 0.1283976584672928 | Loss: 0.0257982634430906 | Epoch: 149 |

MeanAbsoluteError: 0.1304911673069000 | Loss: 0.0254237747793862 | Epoch: 150 |

MeanAbsoluteError: 0.1330068707466125 | Loss: 0.0264143744669855 | Epoch: 151 |

MeanAbsoluteError: 0.1230328232049942 | Loss: 0.0236413720425238 | Epoch: 152 |

MeanAbsoluteError: 0.1269550323486328 | Loss: 0.0255889309786047 | Epoch: 153 |

MeanAbsoluteError: 0.1303141266107559 | Loss: 0.0259740762907313 | Epoch: 154 |

MeanAbsoluteError: 0.1235509589314461 | Loss: 0.0238506868071515 | Epoch: 155 |

MeanAbsoluteError: 0.1253559440374374 | Loss: 0.0245093535417497 | Epoch: 156 |

MeanAbsoluteError: 0.1231924667954445 | Loss: 0.0247702913483954 | Epoch: 157 |

MeanAbsoluteError: 0.1295823603868484 | Loss: 0.0262727898198258 | Epoch: 158 |

MeanAbsoluteError: 0.1278733760118484 | Loss: 0.0251748437492643 | Epoch: 159 |

MeanAbsoluteError: 0.1199144199490547 | Loss: 0.0227695104174200 | Epoch: 160 |

MeanAbsoluteError: 0.1252374053001404 | Loss: 0.0247888246475486 | Epoch: 161 |

MeanAbsoluteError: 0.1227606311440468 | Loss: 0.0233336357145163 | Epoch: 162 |

MeanAbsoluteError: 0.1273369789123535 | Loss: 0.0257185621307387 | Epoch: 163 |

MeanAbsoluteError: 0.1237519830465317 | Loss: 0.0232765136946303 | Epoch: 164 |

MeanAbsoluteError: 0.1229064166545868 | Loss: 0.0237345356884180 | Epoch: 165 |

MeanAbsoluteError: 0.1262762397527695 | Loss: 0.0253981210764808 | Epoch: 166 |

MeanAbsoluteError: 0.1281509995460510 | Loss: 0.0249654590317126 | Epoch: 167 |

MeanAbsoluteError: 0.1239137649536133 | Loss: 0.0242376528247648 | Epoch: 168 |

MeanAbsoluteError: 0.1230630204081535 | Loss: 0.0239877468943208 | Epoch: 169 |

MeanAbsoluteError: 0.1247112378478050 | Loss: 0.0256065924063053 | Epoch: 170 |

MeanAbsoluteError: 0.1229579001665115 | Loss: 0.0249163790212090 | Epoch: 171 |

MeanAbsoluteError: 0.1241895556449890 | Loss: 0.0244219807383827 | Epoch: 172 |

MeanAbsoluteError: 0.1243385672569275 | Loss: 0.0241788867663126 | Epoch: 173 |

MeanAbsoluteError: 0.1253541409969330 | Loss: 0.0248438792563805 | Epoch: 174 |

MeanAbsoluteError: 0.1242505386471748 | Loss: 0.0252868995567527 | Epoch: 175 |

MeanAbsoluteError: 0.1203417703509331 | Loss: 0.0230033636144314 | Epoch: 176 |

MeanAbsoluteError: 0.1245066076517105 | Loss: 0.0246181107750939 | Epoch: 177 |

MeanAbsoluteError: 0.1221471279859543 | Loss: 0.0239063262273946 | Epoch: 178 |

MeanAbsoluteError: 0.1234842538833618 | Loss: 0.0242254837757597 | Epoch: 179 |

MeanAbsoluteError: 0.1209900826215744 | Loss: 0.0230126463896401 | Epoch: 180 |

MeanAbsoluteError: 0.1258654892444611 | Loss: 0.0244968820097468 | Epoch: 181 |

MeanAbsoluteError: 0.1274453699588776 | Loss: 0.0249511652763855 | Epoch: 182 |

MeanAbsoluteError: 0.1274299621582031 | Loss: 0.0260672907218880 | Epoch: 183 |

MeanAbsoluteError: 0.1197483167052269 | Loss: 0.0229019757693944 | Epoch: 184 |

MeanAbsoluteError: 0.1219831183552742 | Loss: 0.0234900968522804 | Epoch: 185 |

MeanAbsoluteError: 0.1258478760719299 | Loss: 0.0243632268416695 | Epoch: 186 |

MeanAbsoluteError: 0.1289843916893005 | Loss: 0.0255433419044130 | Epoch: 187 |

MeanAbsoluteError: 0.1220621392130852 | Loss: 0.0223041181580144 | Epoch: 188 |

MeanAbsoluteError: 0.1205440238118172 | Loss: 0.0223933748658601 | Epoch: 189 |

MeanAbsoluteError: 0.1234936118125916 | Loss: 0.0244739943884391 | Epoch: 190 |

MeanAbsoluteError: 0.1201927065849304 | Loss: 0.0223184843539881 | Epoch: 191 |

MeanAbsoluteError: 0.1359861940145493 | Loss: 0.0281348931079265 | Epoch: 192 |

MeanAbsoluteError: 0.1225375011563301 | Loss: 0.0235323135276364 | Epoch: 193 |

MeanAbsoluteError: 0.1156979724764824 | Loss: 0.0221839884005021 | Epoch: 194 |

MeanAbsoluteError: 0.1230374276638031 | Loss: 0.0237839971993041 | Epoch: 195 |

MeanAbsoluteError: 0.1287959814071655 | Loss: 0.0258700679007840 | Epoch: 196 |

MeanAbsoluteError: 0.1242656931281090 | Loss: 0.0250640546331609 | Epoch: 197 |

MeanAbsoluteError: 0.1216176822781563 | Loss: 0.0225814329611603 | Epoch: 198 |

MeanAbsoluteError: 0.1183160766959190 | Loss: 0.0216939987466321 | Epoch: 199 |

MeanAbsoluteError: 0.1174577325582504 | Loss: 0.0220731549389408 | Epoch: 200 |

MeanAbsoluteError: 0.1178647205233574 | Loss: 0.0218637973417450 | Epoch: 201 |

MeanAbsoluteError: 0.1207670569419861 | Loss: 0.0224049093275971 | Epoch: 202 |

MeanAbsoluteError: 0.1222442239522934 | Loss: 0.0233788612563512 | Epoch: 203 |

MeanAbsoluteError: 0.1254101097583771 | Loss: 0.0239695930171486 | Epoch: 204 |

MeanAbsoluteError: 0.1194647103548050 | Loss: 0.0233337992228917 | Epoch: 205 |

MeanAbsoluteError: 0.1226928457617760 | Loss: 0.0235651551392705 | Epoch: 206 |

MeanAbsoluteError: 0.1238314583897591 | Loss: 0.0238619637972442 | Epoch: 207 |

MeanAbsoluteError: 0.1176069006323814 | Loss: 0.0221152269300849 | Epoch: 208 |

MeanAbsoluteError: 0.1229983121156693 | Loss: 0.0231393391072440 | Epoch: 209 |

MeanAbsoluteError: 0.1220446676015854 | Loss: 0.0229629165236474 | Epoch: 210 |

MeanAbsoluteError: 0.1209176108241081 | Loss: 0.0227465720652253 | Epoch: 211 |

MeanAbsoluteError: 0.1284721046686172 | Loss: 0.0245805704874995 | Epoch: 212 |

MeanAbsoluteError: 0.1285114139318466 | Loss: 0.0248372880018239 | Epoch: 213 |

MeanAbsoluteError: 0.1275059878826141 | Loss: 0.0256342718140998 | Epoch: 214 |

MeanAbsoluteError: 0.1207654848694801 | Loss: 0.0236312665033135 | Epoch: 215 |

MeanAbsoluteError: 0.1192171126604080 | Loss: 0.0224707571422914 | Epoch: 216 |

MeanAbsoluteError: 0.1203290522098541 | Loss: 0.0227653912679186 | Epoch: 217 |

MeanAbsoluteError: 0.1214761883020401 | Loss: 0.0227677798597142 | Epoch: 218 |

MeanAbsoluteError: 0.1233539208769798 | Loss: 0.0233017162022952 | Epoch: 219 |

MeanAbsoluteError: 0.1244300305843353 | Loss: 0.0239834562751154 | Epoch: 220 |

MeanAbsoluteError: 0.1210247576236725 | Loss: 0.0232080408688247 | Epoch: 221 |

MeanAbsoluteError: 0.1149001941084862 | Loss: 0.0209321498541491 | Epoch: 222 |

MeanAbsoluteError: 0.1260302066802979 | Loss: 0.0250623940417184 | Epoch: 223 |

MeanAbsoluteError: 0.1296425759792328 | Loss: 0.0265303449663528 | Epoch: 224 |

MeanAbsoluteError: 0.1263261884450912 | Loss: 0.0234201644613252 | Epoch: 225 |

MeanAbsoluteError: 0.1197481378912926 | Loss: 0.0226027781875261 | Epoch: 226 |

MeanAbsoluteError: 0.1205332800745964 | Loss: 0.0218801511206160 | Epoch: 227 |

MeanAbsoluteError: 0.1243117749691010 | Loss: 0.0235715339456995 | Epoch: 228 |

MeanAbsoluteError: 0.1212009415030479 | Loss: 0.0236190035902352 | Epoch: 229 |

MeanAbsoluteError: 0.1238471493124962 | Loss: 0.0242575536294316 | Epoch: 230 |

MeanAbsoluteError: 0.1184770092368126 | Loss: 0.0239081451293896 | Epoch: 231 |

MeanAbsoluteError: 0.1209292635321617 | Loss: 0.0224181624354484 | Epoch: 232 |

MeanAbsoluteError: 0.1230580285191536 | Loss: 0.0245733254488247 | Epoch: 233 |

MeanAbsoluteError: 0.1239335536956787 | Loss: 0.0228608987199307 | Epoch: 234 |

MeanAbsoluteError: 0.1182504147291183 | Loss: 0.0217470101684254 | Epoch: 235 |

MeanAbsoluteError: 0.1203619986772537 | Loss: 0.0224266062828125 | Epoch: 236 |

MeanAbsoluteError: 0.1190977096557617 | Loss: 0.0228768746853651 | Epoch: 237 |

MeanAbsoluteError: 0.1222894787788391 | Loss: 0.0225844189309282 | Epoch: 238 |

MeanAbsoluteError: 0.1228785216808319 | Loss: 0.0245114814394522 | Epoch: 239 |

MeanAbsoluteError: 0.1206199228763580 | Loss: 0.0224941258190665 | Epoch: 240 |

MeanAbsoluteError: 0.1278737038373947 | Loss: 0.0247227553539172 | Epoch: 241 |

MeanAbsoluteError: 0.1167895644903183 | Loss: 0.0211972226358436 | Epoch: 242 |

MeanAbsoluteError: 0.1184048876166344 | Loss: 0.0216626869243555 | Epoch: 243 |

MeanAbsoluteError: 0.1244337484240532 | Loss: 0.0245967773436132 | Epoch: 244 |

MeanAbsoluteError: 0.1168437972664833 | Loss: 0.0211213659144899 | Epoch: 245 |

MeanAbsoluteError: 0.1190522983670235 | Loss: 0.0220293253432828 | Epoch: 246 |

MeanAbsoluteError: 0.1138084158301353 | Loss: 0.0211509681826283 | Epoch: 247 |

MeanAbsoluteError: 0.1150573864579201 | Loss: 0.0218722552771214 | Epoch: 248 |

MeanAbsoluteError: 0.1180302798748016 | Loss: 0.0220100900266940 | Epoch: 249 |

MeanAbsoluteError: 0.1220537573099136 | Loss: 0.0230908026724258 | Epoch: 250 |

MeanAbsoluteError: 0.1188417300581932 | Loss: 0.0225336065472220 | Epoch: 251 |

MeanAbsoluteError: 0.1138249188661575 | Loss: 0.0200077641700530 | Epoch: 252 |

MeanAbsoluteError: 0.1161041930317879 | Loss: 0.0206555419291059 | Epoch: 253 |

MeanAbsoluteError: 0.1129278019070625 | Loss: 0.0196378064693029 | Epoch: 254 |

MeanAbsoluteError: 0.1191969439387321 | Loss: 0.0217842380475486 | Epoch: 255 |

MeanAbsoluteError: 0.1178955957293510 | Loss: 0.0217673723236658 | Epoch: 256 |

MeanAbsoluteError: 0.1225695312023163 | Loss: 0.0235642511541179 | Epoch: 257 |

MeanAbsoluteError: 0.1189354062080383 | Loss: 0.0221451033592651 | Epoch: 258 |

MeanAbsoluteError: 0.1159379482269287 | Loss: 0.0209394718445643 | Epoch: 259 |

MeanAbsoluteError: 0.1145046129822731 | Loss: 0.0207393894296062 | Epoch: 260 |

MeanAbsoluteError: 0.1135144531726837 | Loss: 0.0209288315546170 | Epoch: 261 |

MeanAbsoluteError: 0.1143190860748291 | Loss: 0.0212320348639635 | Epoch: 262 |

MeanAbsoluteError: 0.1203311905264854 | Loss: 0.0223308805727963 | Epoch: 263 |

MeanAbsoluteError: 0.1187510788440704 | Loss: 0.0223521757698230 | Epoch: 264 |

MeanAbsoluteError: 0.1112951263785362 | Loss: 0.0194042576799984 | Epoch: 265 |

MeanAbsoluteError: 0.1200256124138832 | Loss: 0.0220208585454384 | Epoch: 266 |

MeanAbsoluteError: 0.1108825802803040 | Loss: 0.0195892313580650 | Epoch: 267 |

MeanAbsoluteError: 0.1140665039420128 | Loss: 0.0212596434280082 | Epoch: 268 |

MeanAbsoluteError: 0.1138576492667198 | Loss: 0.0208727094482553 | Epoch: 269 |

MeanAbsoluteError: 0.1228623837232590 | Loss: 0.0229354494923246 | Epoch: 270 |

MeanAbsoluteError: 0.1239947527647018 | Loss: 0.0235173111618496 | Epoch: 271 |

MeanAbsoluteError: 0.1184322088956833 | Loss: 0.0215539103346722 | Epoch: 272 |

MeanAbsoluteError: 0.1139651089906693 | Loss: 0.0196214106114424 | Epoch: 273 |

MeanAbsoluteError: 0.1161968633532524 | Loss: 0.0205856080231024 | Epoch: 274 |

MeanAbsoluteError: 0.1189232617616653 | Loss: 0.0228102921399598 | Epoch: 275 |

MeanAbsoluteError: 0.1183187365531921 | Loss: 0.0211991970293457 | Epoch: 276 |

MeanAbsoluteError: 0.1199806481599808 | Loss: 0.0214618806982374 | Epoch: 277 |

MeanAbsoluteError: 0.1160579398274422 | Loss: 0.0213208495988996 | Epoch: 278 |

MeanAbsoluteError: 0.1179701387882233 | Loss: 0.0215897889621556 | Epoch: 279 |

MeanAbsoluteError: 0.1227130219340324 | Loss: 0.0238527893999011 | Epoch: 280 |

MeanAbsoluteError: 0.1229173764586449 | Loss: 0.0231404031535688 | Epoch: 281 |

MeanAbsoluteError: 0.1178055182099342 | Loss: 0.0214884666224680 | Epoch: 282 |

MeanAbsoluteError: 0.1166307330131531 | Loss: 0.0213225198407114 | Epoch: 283 |

MeanAbsoluteError: 0.1184815838932991 | Loss: 0.0224593540875135 | Epoch: 284 |

MeanAbsoluteError: 0.1215936392545700 | Loss: 0.0226987398487593 | Epoch: 285 |

MeanAbsoluteError: 0.1152554228901863 | Loss: 0.0212227753332506 | Epoch: 286 |

MeanAbsoluteError: 0.1137505620718002 | Loss: 0.0205230907607135 | Epoch: 287 |

MeanAbsoluteError: 0.1227038577198982 | Loss: 0.0233315011995728 | Epoch: 288 |

MeanAbsoluteError: 0.1172978356480598 | Loss: 0.0216168211314653 | Epoch: 289 |

MeanAbsoluteError: 0.1165268421173096 | Loss: 0.0219216217240319 | Epoch: 290 |

MeanAbsoluteError: 0.1112872809171677 | Loss: 0.0201413721899735 | Epoch: 291 |

MeanAbsoluteError: 0.1169391870498657 | Loss: 0.0211478150009740 | Epoch: 292 |

MeanAbsoluteError: 0.1087261065840721 | Loss: 0.0187534253972990 | Epoch: 293 |

MeanAbsoluteError: 0.1149894446134567 | Loss: 0.0205256762765930 | Epoch: 294 |

MeanAbsoluteError: 0.1132562011480331 | Loss: 0.0200840635438605 | Epoch: 295 |

MeanAbsoluteError: 0.1164590567350388 | Loss: 0.0211450988286621 | Epoch: 296 |

MeanAbsoluteError: 0.1234339922666550 | Loss: 0.0227225621935456 | Epoch: 297 |

MeanAbsoluteError: 0.1213747635483742 | Loss: 0.0218261067468363 | Epoch: 298 |

MeanAbsoluteError: 0.1127093881368637 | Loss: 0.0206090404669521 | Epoch: 299 |

MeanAbsoluteError: 0.1139312908053398 | Loss: 0.0207886384594409 | Epoch: 300 |

MeanAbsoluteError: 0.1185982152819633 | Loss: 0.0210673390916781 | Epoch: 301 |

MeanAbsoluteError: 0.1100021600723267 | Loss: 0.0195816084379718 | Epoch: 302 |

MeanAbsoluteError: 0.1146771907806396 | Loss: 0.0209409156637654 | Epoch: 303 |

MeanAbsoluteError: 0.1114989444613457 | Loss: 0.0204456535896558 | Epoch: 304 |

MeanAbsoluteError: 0.1060042157769203 | Loss: 0.0182423085901731 | Epoch: 305 |

MeanAbsoluteError: 0.1107240915298462 | Loss: 0.0195195720785220 | Epoch: 306 |

MeanAbsoluteError: 0.1152724325656891 | Loss: 0.0207731267367975 | Epoch: 307 |

MeanAbsoluteError: 0.1147233545780182 | Loss: 0.0200741056437255 | Epoch: 308 |

MeanAbsoluteError: 0.1116947531700134 | Loss: 0.0197723196629280 |

Epoch: 309 |

MeanAbsoluteError: 0.1184227392077446 | Loss: 0.0224489457386274 | Epoch: 310 |

MeanAbsoluteError: 0.1177092269062996 | Loss: 0.0216322571821608 | Epoch: 311 |

MeanAbsoluteError: 0.1190647482872009 | Loss: 0.0211090057645924 | Epoch: 312 |

MeanAbsoluteError: 0.1083347573876381 | Loss: 0.0184748093901483 | Epoch: 313 |

MeanAbsoluteError: 0.1128053441643715 | Loss: 0.0201024182817491 | Epoch: 314 |

MeanAbsoluteError: 0.1107850670814514 | Loss: 0.0195984465022048 | Epoch: 315 |

MeanAbsoluteError: 0.1091450899839401 | Loss: 0.0194626172037533 | Epoch: 316 |

MeanAbsoluteError: 0.1038403734564781 | Loss: 0.0178249661163697 | Epoch: 317 |

MeanAbsoluteError: 0.1171389669179916 | Loss: 0.0213841378810654 | Epoch: 318 |

MeanAbsoluteError: 0.1109480038285255 | Loss: 0.0190781150546779 | Epoch: 319 |

MeanAbsoluteError: 0.1136367172002792 | Loss: 0.0201267168830236 | Epoch: 320 |

MeanAbsoluteError: 0.1172280088067055 | Loss: 0.0222621215877977 | Epoch: 321 |

MeanAbsoluteError: 0.1155606955289841 | Loss: 0.0203938189101367 | Epoch: 322 |

MeanAbsoluteError: 0.1099353805184364 | Loss: 0.0197518191332347 | Epoch: 323 |

MeanAbsoluteError: 0.1141984686255455 | Loss: 0.0202540502073528 | Epoch: 324 |

MeanAbsoluteError: 0.1124569475650787 | Loss: 0.0196308325407639 | Epoch: 325 |

MeanAbsoluteError: 0.1116140782833099 | Loss: 0.0194612937883358 | Epoch: 326 |

MeanAbsoluteError: 0.1172863021492958 | Loss: 0.0207787396640651 | Epoch: 327 |

MeanAbsoluteError: 0.1136894226074219 | Loss: 0.0197656322567491 | Epoch: 328 |

MeanAbsoluteError: 0.1135412752628326 | Loss: 0.0206906095358621 | Epoch: 329 |

MeanAbsoluteError: 0.1099154949188232 | Loss: 0.0195675870151839 | Epoch: 330 |

MeanAbsoluteError: 0.1114951446652412 | Loss: 0.0194799950518548 | Epoch: 331 |

MeanAbsoluteError: 0.1203667297959328 | Loss: 0.0221433327442113 | Epoch: 332 |

MeanAbsoluteError: 0.1172201782464981 | Loss: 0.0214955961201728 | Epoch: 333 |

MeanAbsoluteError: 0.1101076230406761 | Loss: 0.0190324027752195 | Epoch: 334 |

MeanAbsoluteError: 0.1202044039964676 | Loss: 0.0217025208938867 | Epoch: 335 |

MeanAbsoluteError: 0.1143727004528046 | Loss: 0.0203296812928844 | Epoch: 336 |

MeanAbsoluteError: 0.1074640005826950 | Loss: 0.0183476901652345 | Epoch: 337 |

MeanAbsoluteError: 0.1090474724769592 | Loss: 0.0196792662288256 | Epoch: 338 |

MeanAbsoluteError: 0.1105525046586990 | Loss: 0.0194957763344185 | Epoch: 339 |

MeanAbsoluteError: 0.1099732816219330 | Loss: 0.0199719384095321 | Epoch: 340 |

MeanAbsoluteError: 0.1112416237592697 | Loss: 0.0185751687509279 | Epoch: 341 |

MeanAbsoluteError: 0.1115840375423431 | Loss: 0.0185860737370482 | Epoch: 342 |

MeanAbsoluteError: 0.1100447699427605 | Loss: 0.0185696589054229 | Epoch: 343 |

MeanAbsoluteError: 0.1146142706274986 | Loss: 0.0205673177303591 | Epoch: 344 |

MeanAbsoluteError: 0.1180334240198135 | Loss: 0.0215321239770856 | Epoch: 345 |

MeanAbsoluteError: 0.1154225170612335 | Loss: 0.0203762664787549 | Epoch: 346 |

MeanAbsoluteError: 0.1113865002989769 | Loss: 0.0198626081445642 | Epoch: 347 |

MeanAbsoluteError: 0.1113539114594460 | Loss: 0.0185956120409537 | Epoch: 348 |

MeanAbsoluteError: 0.1109112277626991 | Loss: 0.0204713044206437 | Epoch: 349 |

MeanAbsoluteError: 0.1102373898029327 | Loss: 0.0183471426501637 | Epoch: 350 |

MeanAbsoluteError: 0.1079941317439079 | Loss: 0.0178741592704925 | Epoch: 351 |

MeanAbsoluteError: 0.1133004501461983 | Loss: 0.0195486071342990 | Epoch: 352 |

MeanAbsoluteError: 0.1091309711337090 | Loss: 0.0192581023531966 | Epoch: 353 |

MeanAbsoluteError: 0.1106746271252632 | Loss: 0.0197598629271670 | Epoch: 354 |

MeanAbsoluteError: 0.1061421483755112 | Loss: 0.0183699913645008 | Epoch: 355 |

MeanAbsoluteError: 0.1197916194796562 | Loss: 0.0237493675776447 | Epoch: 356 |

MeanAbsoluteError: 0.1131652444601059 | Loss: 0.0199773004423817 | Epoch: 357 |

MeanAbsoluteError: 0.1060523763298988 | Loss: 0.0176641216907107 | Epoch: 358 |

MeanAbsoluteError: 0.1121506020426750 | Loss: 0.0201430348924381 | Epoch: 359 |

MeanAbsoluteError: 0.1088513806462288 | Loss: 0.0181910248589944 | Epoch: 360 |

MeanAbsoluteError: 0.1114470958709717 | Loss: 0.0202714130242627 | Epoch: 361 |

MeanAbsoluteError: 0.1084359213709831 | Loss: 0.0181774812148069 | Epoch: 362 |

MeanAbsoluteError: 0.1131586357951164 | Loss: 0.0202884888344367 | Epoch: 363 |

MeanAbsoluteError: 0.1085071936249733 | Loss: 0.0186360509012593 | Epoch: 364 |

MeanAbsoluteError: 0.1097636669874191 | Loss: 0.0193709138991350 | Epoch: 365 |

MeanAbsoluteError: 0.1120966821908951 | Loss: 0.0196423731649217 | Epoch: 366 |

MeanAbsoluteError: 0.1163576766848564 | Loss: 0.0211185560436570 | Epoch: 367 |

MeanAbsoluteError: 0.1069724559783936 | Loss: 0.0182413802355101 | Epoch: 368 |

MeanAbsoluteError: 0.1158574447035789 | Loss: 0.0215859119286082 | Epoch: 369 |

MeanAbsoluteError: 0.1151331961154938 | Loss: 0.0217689722171053 | Epoch: 370 |

MeanAbsoluteError: 0.1170153692364693 | Loss: 0.0213258852182965 | Epoch: 371 |

MeanAbsoluteError: 0.1130115613341331 | Loss: 0.0207253134681378 | Epoch: 372 |

MeanAbsoluteError: 0.1014262735843658 | Loss: 0.0172444242029451 | Epoch: 373 |

MeanAbsoluteError: 0.1079891100525856 | Loss: 0.0187004893554240 | Epoch: 374 |

MeanAbsoluteError: 0.1104081496596336 | Loss: 0.0190538861916866 | Epoch: 375 |

MeanAbsoluteError: 0.1107673421502113 | Loss: 0.0193246308959594 | Epoch: 376 |

MeanAbsoluteError: 0.1056442558765411 | Loss: 0.0185647574392654 | Epoch: 377 |

MeanAbsoluteError: 0.1040799021720886 | Loss: 0.0178372529238307 | Epoch: 378 |

MeanAbsoluteError: 0.1148910671472549 | Loss: 0.0208913113199136 | Epoch: 379 |

MeanAbsoluteError: 0.1141818091273308 | Loss: 0.0201505185857726 | Epoch: 380 |

MeanAbsoluteError: 0.1101194769144058 | Loss: 0.0184993584140223 | Epoch: 381 |

MeanAbsoluteError: 0.1112821325659752 | Loss: 0.0191732752729877 | Epoch: 382 |

MeanAbsoluteError: 0.1083045080304146 | Loss: 0.0184835520648024 | Epoch: 383 |

MeanAbsoluteError: 0.1031277924776077 | Loss: 0.0176558217865992 | Epoch: 384 |

MeanAbsoluteError: 0.1162542626261711 | Loss: 0.0206359932509561 | Epoch: 385 |

MeanAbsoluteError: 0.1100053116679192 | Loss: 0.0186747396927482 | Epoch: 386 |

MeanAbsoluteError: 0.1042319014668465 | Loss: 0.0172732980792838 | Epoch: 387 |

MeanAbsoluteError: 0.1054554954171181 | Loss: 0.0179135237121469 | Epoch: 388 |

MeanAbsoluteError: 0.1136918142437935 | Loss: 0.0205074256897691 | Epoch: 389 |

MeanAbsoluteError: 0.1144536435604095 | Loss: 0.0207948570311419 | Epoch: 390 |

MeanAbsoluteError: 0.1063778251409531 | Loss: 0.0184382278054788 | Epoch: 391 |

MeanAbsoluteError: 0.1093017458915710 | Loss: 0.0189151251932102 | Epoch: 392 |

MeanAbsoluteError: 0.1103127673268318 | Loss: 0.0189185671649708 | Epoch: 393 |

MeanAbsoluteError: 0.1163487210869789 | Loss: 0.0204652114893058 | Epoch: 394 |

MeanAbsoluteError: 0.1059659570455551 | Loss: 0.0175768491303218 | Epoch: 395 |

MeanAbsoluteError: 0.1061762496829033 | Loss: 0.0181319024823764 | Epoch: 396 |

MeanAbsoluteError: 0.1082430258393288 | Loss: 0.0174185658502392 | Epoch: 397 |

MeanAbsoluteError: 0.1135094985365868 | Loss: 0.0197790348138369 | Epoch: 398 |

MeanAbsoluteError: 0.1118199303746223 | Loss: 0.0194046005930674 | Epoch: 399 |

MeanAbsoluteError: 0.1096143499016762 | Loss: 0.0186319853877406 | Epoch: 400 |

MeanAbsoluteError: 0.1098315417766571 | Loss: 0.0194485069151627 | Epoch: 401 |

MeanAbsoluteError: 0.1078771948814392 | Loss: 0.0183863855820285 | Epoch: 402 |

MeanAbsoluteError: 0.0985797420144081 | Loss: 0.0162425671764019 | Epoch: 403 |

MeanAbsoluteError: 0.1073623374104500 | Loss: 0.0179628877468834 | Epoch: 404 |

MeanAbsoluteError: 0.1028630584478378 | Loss: 0.0178884074802530 | Epoch: 405 |

MeanAbsoluteError: 0.1082184761762619 | Loss: 0.0183466988605990 | Epoch: 406 |

MeanAbsoluteError: 0.1104218661785126 | Loss: 0.0186570576659869 | Epoch: 407 |

MeanAbsoluteError: 0.1006022244691849 | Loss: 0.0165893402691775 | Epoch: 408 |

MeanAbsoluteError: 0.1080789044499397 | Loss: 0.0184906883639148 | Epoch: 409 |

MeanAbsoluteError: 0.1065803915262222 | Loss: 0.0179645125131841 | Epoch: 410 |

MeanAbsoluteError: 0.1124235391616821 | Loss: 0.0203065908331579 | Epoch: 411 |

MeanAbsoluteError: 0.1093000248074532 | Loss: 0.0191671217917368 | Epoch: 412 |

MeanAbsoluteError: 0.0966078862547874 | Loss: 0.0152736370590342 | Epoch: 413 |

MeanAbsoluteError: 0.1038100868463516 | Loss: 0.0168429692335970 | Epoch: 414 |

MeanAbsoluteError: 0.1072236746549606 | Loss: 0.0186142792037572 | Epoch: 415 |

MeanAbsoluteError: 0.1052646860480309 | Loss: 0.0174917519066366 | Epoch: 416 |

MeanAbsoluteError: 0.1088057681918144 | Loss: 0.0178884330691653 | Epoch: 417 |

MeanAbsoluteError: 0.1104790866374969 | Loss: 0.0196447814899147 | Epoch: 418 |

MeanAbsoluteError: 0.1005770936608315 | Loss: 0.0154605810453844 | Epoch: 419 |

MeanAbsoluteError: 0.1070555001497269 | Loss: 0.0177700087717191 | Epoch: 420 |

MeanAbsoluteError: 0.1097387820482254 | Loss: 0.0189863474376034 | Epoch: 421 |

MeanAbsoluteError: 0.1024144440889359 | Loss: 0.0170932739910495 | Epoch: 422 |

MeanAbsoluteError: 0.1085557937622070 | Loss: 0.0188421526051631 | Epoch: 423 |

MeanAbsoluteError: 0.1129229366779327 | Loss: 0.0197384734790921 | Epoch: 424 |

MeanAbsoluteError: 0.1020447090268135 | Loss: 0.0154831843953192 | Epoch: 425 |

MeanAbsoluteError: 0.0974431484937668 | Loss: 0.0144806950667165 | Epoch: 426 |

MeanAbsoluteError: 0.1066611930727959 | Loss: 0.0182443831500132 | Epoch: 427 |

MeanAbsoluteError: 0.1082078516483307 | Loss: 0.0186480926018339 | Epoch: 428 |

MeanAbsoluteError: 0.1127255484461784 | Loss: 0.0199842765862316 | Epoch: 429 |

MeanAbsoluteError: 0.1010202616453171 | Loss: 0.0157428803886554 | Epoch: 430 |

MeanAbsoluteError: 0.1073424816131592 | Loss: 0.0184276818295863 | Epoch: 431 |

MeanAbsoluteError: 0.1078823879361153 | Loss: 0.0189910158392740 | Epoch: 432 |

MeanAbsoluteError: 0.1061350405216217 | Loss: 0.0182186201614483 | Epoch: 433 |

MeanAbsoluteError: 0.1050368621945381 | Loss: 0.0174774029051089 | Epoch: 434 |

MeanAbsoluteError: 0.1022097021341324 | Loss: 0.0176141184859898 | Epoch: 435 |

MeanAbsoluteError: 0.1035732775926590 | Loss: 0.0170162531867815 | Epoch: 436 |

MeanAbsoluteError: 0.1107955649495125 | Loss: 0.0189139174593341 | Epoch: 437 |

MeanAbsoluteError: 0.1076001003384590 | Loss: 0.0181775151464778 | Epoch: 438 |

MeanAbsoluteError: 0.1103157550096512 | Loss: 0.0180585617210212 | Epoch: 439 |

MeanAbsoluteError: 0.0987242907285690 | Loss: 0.0166644142848478 | Epoch: 440 |

MeanAbsoluteError: 0.1042128056287766 | Loss: 0.0178156605951032 | Epoch: 441 |

MeanAbsoluteError: 0.1036315336823463 | Loss: 0.0166473287887250 | Epoch: 442 |

MeanAbsoluteError: 0.1042812764644623 | Loss: 0.0168682512188874 | Epoch: 443 |

MeanAbsoluteError: 0.1039046868681908 | Loss: 0.0169052274274388 | Epoch: 444 |

MeanAbsoluteError: 0.1005613729357719 | Loss: 0.0171008928685236 | Epoch: 445 |

MeanAbsoluteError: 0.1089297086000443 | Loss: 0.0185141705477387 | Epoch: 446 |

MeanAbsoluteError: 0.1029682159423828 | Loss: 0.0170057015290270 | Epoch: 447 |

MeanAbsoluteError: 0.1019812151789665 | Loss: 0.0172717804495763 | Epoch: 448 |

MeanAbsoluteError: 0.1016053408384323 | Loss: 0.0168519639029788 | Epoch: 449 |

MeanAbsoluteError: 0.1032952666282654 | Loss: 0.0174556447161497 | Epoch: 450 |

MeanAbsoluteError: 0.1090637966990471 | Loss: 0.0194075546173553 | Epoch: 451 |

MeanAbsoluteError: 0.0968510285019875 | Loss: 0.0153647898138782 | Epoch: 452 |

MeanAbsoluteError: 0.1080319434404373 | Loss: 0.0181882659715605 | Epoch: 453 |

MeanAbsoluteError: 0.1026531010866165 | Loss: 0.0165582563802794 | Epoch: 454 |

MeanAbsoluteError: 0.1027656644582748 | Loss: 0.0169871264434187 | Epoch: 455 |

MeanAbsoluteError: 0.1030584499239922 | Loss: 0.0182213424418417 | Epoch: 456 |

MeanAbsoluteError: 0.1050972566008568 | Loss: 0.0176845944045878 | Epoch: 457 |

MeanAbsoluteError: 0.1082886606454849 | Loss: 0.0188664381343794 | Epoch: 458 |

MeanAbsoluteError: 0.1030610427260399 | Loss: 0.0168680448738936 | Epoch: 459 |

MeanAbsoluteError: 0.1062390878796577 | Loss: 0.0176622357205391 | Epoch: 460 |

MeanAbsoluteError: 0.1010513603687286 | Loss: 0.0160592330798924 | Epoch: 461 |

MeanAbsoluteError: 0.1037242189049721 | Loss: 0.0175088923414781 | Epoch: 462 |

MeanAbsoluteError: 0.0996847227215767 | Loss: 0.0162871328825895 | Epoch: 463 |

MeanAbsoluteError: 0.1028450280427933 | Loss: 0.0175891913717351 | Epoch: 464 |

MeanAbsoluteError: 0.1076999828219414 | Loss: 0.0182758360010727 | Epoch: 465 |

MeanAbsoluteError: 0.0998352468013763 | Loss: 0.0156349182651805 | Epoch: 466 |

MeanAbsoluteError: 0.0980077013373375 | Loss: 0.0154706906826686 | Epoch: 467 |

MeanAbsoluteError: 0.0983856990933418 | Loss: 0.0155107726874606 | Epoch: 468 |

MeanAbsoluteError: 0.1016805022954941 | Loss: 0.0177262570226837 | Epoch: 469 |

MeanAbsoluteError: 0.1044908687472343 | Loss: 0.0180568924769129 | Epoch: 470 |

MeanAbsoluteError: 0.1044169664382935 | Loss: 0.0180222642709001 | Epoch: 471 |

MeanAbsoluteError: 0.0998695790767670 | Loss: 0.0173250358633716 | Epoch: 472 |

MeanAbsoluteError: 0.1001142412424088 | Loss: 0.0167369568090847 | Epoch: 473 |

MeanAbsoluteError: 0.1080079749226570 | Loss: 0.0179532358421905 | Epoch: 474 |

MeanAbsoluteError: 0.1014707535505295 | Loss: 0.0168618322504835 | Epoch: 475 |

MeanAbsoluteError: 0.1033758297562599 | Loss: 0.0171427309479138 | Epoch: 476 |

MeanAbsoluteError: 0.1050861328840256 | Loss: 0.0179493298879243 | Epoch: 477 |

MeanAbsoluteError: 0.0991537272930145 | Loss: 0.0155396374180176 | Epoch: 478 |

MeanAbsoluteError: 0.0994060561060905 | Loss: 0.0162475738374633 | Epoch: 479 |

MeanAbsoluteError: 0.1072624996304512 | Loss: 0.0181582321104967 | Epoch: 480 |

MeanAbsoluteError: 0.1012874171137810 | Loss: 0.0174265437384990 | Epoch: 481 |

MeanAbsoluteError: 0.1009065657854080 | Loss: 0.0159158798698869 | Epoch: 482 |

MeanAbsoluteError: 0.1054950878024101 | Loss: 0.0181722106381009 | Epoch: 483 |

MeanAbsoluteError: 0.0971867144107819 | Loss: 0.0156648227361438 | Epoch: 484 |

MeanAbsoluteError: 0.1049359813332558 | Loss: 0.0172289869571978 | Epoch: 485 |

MeanAbsoluteError: 0.1019908413290977 | Loss: 0.0170625475561246 | Epoch: 486 |

MeanAbsoluteError: 0.1018972173333168 | Loss: 0.0163799107017257 | Epoch: 487 |

MeanAbsoluteError: 0.0923860073089600 | Loss: 0.0140937910763508 | Epoch: 488 |

MeanAbsoluteError: 0.0976248905062675 | Loss: 0.0155754736408320 | Epoch: 489 |

MeanAbsoluteError: 0.1014536321163177 | Loss: 0.0166061183147516 | Epoch: 490 |

MeanAbsoluteError: 0.1052436381578445 | Loss: 0.0184943631670224 | Epoch: 491 |

MeanAbsoluteError: 0.0991752594709396 | Loss: 0.0156068176250362 | Epoch: 492 |

MeanAbsoluteError: 0.0947347655892372 | Loss: 0.0148993021188411 | Epoch: 493 |

MeanAbsoluteError: 0.1100518777966499 | Loss: 0.0191614920042533 | Epoch: 494 |

MeanAbsoluteError: 0.1056070923805237 | Loss: 0.0173845929727152 | Epoch: 495 |

MeanAbsoluteError: 0.1005499660968781 | Loss: 0.0164102801292514 | Epoch: 496 |

MeanAbsoluteError: 0.1027273982763290 | Loss: 0.0165491457164292 | Epoch: 497 |

MeanAbsoluteError: 0.0989020764827728 | Loss: 0.0170743581438031 | Epoch: 498 |

MeanAbsoluteError: 0.1039193496108055 | Loss: 0.0183461209881110 | Epoch: 499 |

MeanAbsoluteError: 0.1026449576020241 | Loss: 0.0168292134146517 | Epoch: 500 |

MeanAbsoluteError: 0.0939760431647301 | Loss: 0.0147978980380382 | Epoch: 501 |

MeanAbsoluteError: 0.1077166125178337 | Loss: 0.0183066413398774 | Epoch: 502 |

MeanAbsoluteError: 0.0993370488286018 | Loss: 0.0163019097216844 | Epoch: 503 |

MeanAbsoluteError: 0.0945171266794205 | Loss: 0.0153303724879515 | Epoch: 504 |

MeanAbsoluteError: 0.1008688509464264 | Loss: 0.0167775752919260 | Epoch: 505 |

MeanAbsoluteError: 0.1000161617994308 | Loss: 0.0160814810981780 | Epoch: 506 |

MeanAbsoluteError: 0.0964475125074387 | Loss: 0.0152281972941516 | Epoch: 507 |

MeanAbsoluteError: 0.0980840399861336 | Loss: 0.0151069655128716 | Epoch: 508 |

MeanAbsoluteError: 0.1020107269287109 | Loss: 0.0166987450830250 | Epoch: 509 |

MeanAbsoluteError: 0.1021979674696922 | Loss: 0.0167664950191465 | Epoch: 510 |

MeanAbsoluteError: 0.0917973965406418 | Loss: 0.0138971836048950 | Epoch: 511 |

MeanAbsoluteError: 0.0963429212570190 | Loss: 0.0152386311171881 | Epoch: 512 |

MeanAbsoluteError: 0.1007172763347626 | Loss: 0.0163271431545976 | Epoch: 513 |

MeanAbsoluteError: 0.0974454432725906 | Loss: 0.0150090079839962 | Epoch: 514 |

MeanAbsoluteError: 0.0970909595489502 | Loss: 0.0152078219187570 | Epoch: 515 |

MeanAbsoluteError: 0.1093857213854790 | Loss: 0.0189685863059033 | Epoch: 516 |

MeanAbsoluteError: 0.1001348719000816 | Loss: 0.0159167319953364 | Epoch: 517 |

MeanAbsoluteError: 0.1002429872751236 | Loss: 0.0173474698936722 | Epoch: 518 |

MeanAbsoluteError: 0.0968904420733452 | Loss: 0.0147789504478290 | Epoch: 519 |

MeanAbsoluteError: 0.1099022030830383 | Loss: 0.0189467534753688 | Epoch: 520 |

MeanAbsoluteError: 0.0999557301402092 | Loss: 0.0163296968406727 | Epoch: 521 |

MeanAbsoluteError: 0.0940432772040367 | Loss: 0.0150107734339629 | Epoch: 522 |

MeanAbsoluteError: 0.1007972657680511 | Loss: 0.0165020587223019 | Epoch: 523 |

MeanAbsoluteError: 0.1039046943187714 | Loss: 0.0179870286903558 | Epoch: 524 |

MeanAbsoluteError: 0.1025696396827698 | Loss: 0.0168178719457258 | Epoch: 525 |

MeanAbsoluteError: 0.0984179973602295 | Loss: 0.0154235302003993 | Epoch: 526 |

MeanAbsoluteError: 0.0999756827950478 | Loss: 0.0161285601375372 | Epoch: 527 |

MeanAbsoluteError: 0.1014905944466591 | Loss: 0.0169098500771361 | Epoch: 528 |

MeanAbsoluteError: 0.0993648767471313 | Loss: 0.0157294345559058 | Epoch: 529 |

MeanAbsoluteError: 0.0958481952548027 | Loss: 0.0151875809337071 | Epoch: 530 |

MeanAbsoluteError: 0.0981381163001060 | Loss: 0.0157585828482053 | Epoch: 531 |

MeanAbsoluteError: 0.0965672805905342 | Loss: 0.0149766461527421 | Epoch: 532 |

MeanAbsoluteError: 0.0964750424027443 | Loss: 0.0154436523162440 | Epoch: 533 |

MeanAbsoluteError: 0.0996313169598579 | Loss: 0.0153808052581735 | Epoch: 534 |

MeanAbsoluteError: 0.1016926169395447 | Loss: 0.0164642078203421 | Epoch: 535 |

MeanAbsoluteError: 0.0961120948195457 | Loss: 0.0155447739682859 | Epoch: 536 |

MeanAbsoluteError: 0.1009661704301834 | Loss: 0.0154702615079016 | Epoch: 537 |

MeanAbsoluteError: 0.0927701070904732 | Loss: 0.0139665581687101 | Epoch: 538 |

MeanAbsoluteError: 0.0975929275155067 | Loss: 0.0158387144066364 | Epoch: 539 |

MeanAbsoluteError: 0.1090823113918304 | Loss: 0.0187838930328508 | Epoch: 540 |

MeanAbsoluteError: 0.0926862731575966 | Loss: 0.0139182428712957 | Epoch: 541 |

MeanAbsoluteError: 0.0952141359448433 | Loss: 0.0151013247385466 | Epoch: 542 |

MeanAbsoluteError: 0.1026523187756538 | Loss: 0.0169537403314704 | Epoch: 543 |

MeanAbsoluteError: 0.0927795246243477 | Loss: 0.0137619458422220 | Epoch: 544 |

MeanAbsoluteError: 0.1018760204315186 | Loss: 0.0173547741950218 | Epoch: 545 |

MeanAbsoluteError: 0.0876221582293510 | Loss: 0.0129409775619085 | Epoch: 546 |

MeanAbsoluteError: 0.0948479548096657 | Loss: 0.0155146177764254 | Epoch: 547 |

MeanAbsoluteError: 0.0949814319610596 | Loss: 0.0144324240290734 | Epoch: 548 |

MeanAbsoluteError: 0.1003346219658852 | Loss: 0.0164650584347934 | Epoch: 549 |

MeanAbsoluteError: 0.0971016734838486 | Loss: 0.0160147672195308 | Epoch: 550 |

MeanAbsoluteError: 0.0968489944934845 | Loss: 0.0155255982812378 | Epoch: 551 |

MeanAbsoluteError: 0.0989157184958458 | Loss: 0.0159895595274672 | Epoch: 552 |

MeanAbsoluteError: 0.0949094891548157 | Loss: 0.0159155225827756 | Epoch: 553 |

MeanAbsoluteError: 0.0996364429593086 | Loss: 0.0171524579235605 | Epoch: 554 |

MeanAbsoluteError: 0.0945355296134949 | Loss: 0.0136509586978354 | Epoch: 555 |

MeanAbsoluteError: 0.0961039140820503 | Loss: 0.0149329183692074 | Epoch: 556 |

MeanAbsoluteError: 0.0990012139081955 | Loss: 0.0165087454026313 | Epoch: 557 |

MeanAbsoluteError: 0.0989302769303322 | Loss: 0.0161787420539379 | Epoch: 558 |

MeanAbsoluteError: 0.0916829332709312 | Loss: 0.0142824590044135 | Epoch: 559 |

MeanAbsoluteError: 0.0980954542756081 | Loss: 0.0163639789991430 | Epoch: 560 |

MeanAbsoluteError: 0.0932519435882568 | Loss: 0.0141505422331587 | Epoch: 561 |

MeanAbsoluteError: 0.0952534526586533 | Loss: 0.0145549110552626 | Epoch: 562 |

MeanAbsoluteError: 0.0996774584054947 | Loss: 0.0156302000666619 | Epoch: 563 |

MeanAbsoluteError: 0.0977452695369720 | Loss: 0.0161742547542478 | Epoch: 564 |

MeanAbsoluteError: 0.0895439311861992 | Loss: 0.0132933590046984 | Epoch: 565 |

MeanAbsoluteError: 0.0941376537084579 | Loss: 0.0146976954252265 | Epoch: 566 |

MeanAbsoluteError: 0.1013995260000229 | Loss: 0.0165438991334910 | Epoch: 567 |

MeanAbsoluteError: 0.1030474603176117 | Loss: 0.0171867421689209 | Epoch: 568 |

MeanAbsoluteError: 0.0905660465359688 | Loss: 0.0138430645086676 | Epoch: 569 |

MeanAbsoluteError: 0.0905057564377785 | Loss: 0.0134842142437022 | Epoch: 570 |

MeanAbsoluteError: 0.0910122320055962 | Loss: 0.0136786883624639 | Epoch: 571 |

MeanAbsoluteError: 0.0933640524744987 | Loss: 0.0150762352257637 | Epoch: 572 |

MeanAbsoluteError: 0.0958045050501823 | Loss: 0.0152325371302868 | Epoch: 573 |

MeanAbsoluteError: 0.0984153971076012 | Loss: 0.0157607538873466 | Epoch: 574 |

MeanAbsoluteError: 0.0905474945902824 | Loss: 0.0135822923557619 | Epoch: 575 |

MeanAbsoluteError: 0.0918791666626930 | Loss: 0.0142797212744154 | Epoch: 576 |

MeanAbsoluteError: 0.0922047346830368 | Loss: 0.0138489314861363 | Epoch: 577 |

MeanAbsoluteError: 0.0958790108561516 | Loss: 0.0150872529657014 | Epoch: 578 |

MeanAbsoluteError: 0.0954126641154289 | Loss: 0.0150977555640566 | Epoch: 579 |

MeanAbsoluteError: 0.1041245311498642 | Loss: 0.0187601747452572 | Epoch: 580 |

MeanAbsoluteError: 0.0959928110241890 | Loss: 0.0142142682177170 | Epoch: 581 |

MeanAbsoluteError: 0.0892215967178345 | Loss: 0.0133904045842549 | Epoch: 582 |

MeanAbsoluteError: 0.0938471630215645 | Loss: 0.0136533047924361 | Epoch: 583 |

MeanAbsoluteError: 0.0891572833061218 | Loss: 0.0131718141755361 | Epoch: 584 |

MeanAbsoluteError: 0.0962666049599648 | Loss: 0.0152779891085083 | Epoch: 585 |

MeanAbsoluteError: 0.0949072614312172 | Loss: 0.0148243514096733 | Epoch: 586 |

MeanAbsoluteError: 0.0947503522038460 | Loss: 0.0140619072997166 | Epoch: 587 |

MeanAbsoluteError: 0.0899440199136734 | Loss: 0.0123618191588563 | Epoch: 588 |

MeanAbsoluteError: 0.0914239287376404 | Loss: 0.0136703018918100 | Epoch: 589 |

MeanAbsoluteError: 0.0943471118807793 | Loss: 0.0145854217026014 | Epoch: 590 |

MeanAbsoluteError: 0.0967638194561005 | Loss: 0.0153680206452069 | Epoch: 591 |

MeanAbsoluteError: 0.0934237316250801 | Loss: 0.0138307518876779 | Epoch: 592 |

MeanAbsoluteError: 0.0914214104413986 | Loss: 0.0138032288863421 | Epoch: 593 |

MeanAbsoluteError: 0.0960265845060349 | Loss: 0.0160195194552701 | Epoch: 594 |

MeanAbsoluteError: 0.0908762589097023 | Loss: 0.0144466645822104 | Epoch: 595 |

MeanAbsoluteError: 0.0973815470933914 | Loss: 0.0152153484319084 | Epoch: 596 |

MeanAbsoluteError: 0.0903173089027405 | Loss: 0.0136349404588206 | Epoch: 597 |

MeanAbsoluteError: 0.0993798300623894 | Loss: 0.0164040038574725 | Epoch: 598 |

MeanAbsoluteError: 0.0928648784756660 | Loss: 0.0148511308150773 | Epoch: 599 |

MeanAbsoluteError: 0.0925984457135201 | Loss: 0.0144162231851078 | Epoch: 600 |

MeanAbsoluteError: 0.0999183505773544 | Loss: 0.0163086755104208 | Epoch: 601 |

MeanAbsoluteError: 0.0949678793549538 | Loss: 0.0147953141444790 | Epoch: 602 |

MeanAbsoluteError: 0.0939765349030495 | Loss: 0.0151060917086215 | Epoch: 603 |

MeanAbsoluteError: 0.0900883451104164 | Loss: 0.0144351810567120 | Epoch: 604 |

MeanAbsoluteError: 0.0943650528788567 | Loss: 0.0142321199094780 | Epoch: 605 |

MeanAbsoluteError: 0.0943175479769707 | Loss: 0.0144278338145705 | Epoch: 606 |

MeanAbsoluteError: 0.0966250672936440 | Loss: 0.0148062422547567 | Epoch: 607 |

MeanAbsoluteError: 0.0929063409566879 | Loss: 0.0143242070393414 | Epoch: 608 |

MeanAbsoluteError: 0.0964447259902954 | Loss: 0.0165878285482177 | Epoch: 609 |

MeanAbsoluteError: 0.0911976322531700 | Loss: 0.0141807622512958 | Epoch: 610 |

MeanAbsoluteError: 0.0998397096991539 | Loss: 0.0168758690922550 | Epoch: 611 |

MeanAbsoluteError: 0.0933445617556572 | Loss: 0.0141533588031598 | Epoch: 612 |

MeanAbsoluteError: 0.1003218889236450 | Loss: 0.0170749911178215 | Epoch: 613 |

MeanAbsoluteError: 0.0928163006901741 | Loss: 0.0134476699212852 | Epoch: 614 |

MeanAbsoluteError: 0.0897209718823433 | Loss: 0.0137047016221428 | Epoch: 615 |

MeanAbsoluteError: 0.0935354456305504 | Loss: 0.0142482936388357 | Epoch: 616 |

MeanAbsoluteError: 0.0933858901262283 | Loss: 0.0135930238121493 | Epoch: 617 |

MeanAbsoluteError: 0.0918898880481720 | Loss: 0.0136457925226114 | Epoch: 618 |

MeanAbsoluteError: 0.0930190011858940 | Loss: 0.0148269979656592 | Epoch: 619 |

MeanAbsoluteError: 0.0951678082346916 | Loss: 0.0146986238623746 | Epoch: 620 |

MeanAbsoluteError: 0.0936522185802460 | Loss: 0.0144980985643148 | Epoch: 621 |

MeanAbsoluteError: 0.0880865082144737 | Loss: 0.0127878839933631 | Epoch: 622 |

MeanAbsoluteError: 0.0964249148964882 | Loss: 0.0150128442532878 | Epoch: 623 |

MeanAbsoluteError: 0.0964215323328972 | Loss: 0.0150659070204711 | Epoch: 624 |

MeanAbsoluteError: 0.0916870385408401 | Loss: 0.0139418501170682 | Epoch: 625 |

MeanAbsoluteError: 0.0943727493286133 | Loss: 0.0147048733378930 | Epoch: 626 |

MeanAbsoluteError: 0.0872936844825745 | Loss: 0.0126948013046422 | Epoch: 627 |

MeanAbsoluteError: 0.0949321165680885 | Loss: 0.0146096022160176 | Epoch: 628 |

MeanAbsoluteError: 0.0940456688404083 | Loss: 0.0145572021667613 | Epoch: 629 |

MeanAbsoluteError: 0.0927180722355843 | Loss: 0.0145954246538652 | Epoch: 630 |

MeanAbsoluteError: 0.0939121916890144 | Loss: 0.0143216857194542 | Epoch: 631 |

MeanAbsoluteError: 0.0899544879794121 | Loss: 0.0141671223289207 | Epoch: 632 |

MeanAbsoluteError: 0.0897123292088509 | Loss: 0.0139130756513138 | Epoch: 633 |

MeanAbsoluteError: 0.0875496789813042 | Loss: 0.0122651119255412 | Epoch: 634 |

MeanAbsoluteError: 0.0856055542826653 | Loss: 0.0122901948574387 | Epoch: 635 |

MeanAbsoluteError: 0.0943021997809410 | Loss: 0.0145512787488406 | Epoch: 636 |

MeanAbsoluteError: 0.0920844003558159 | Loss: 0.0140073012297459 | Epoch: 637 |

MeanAbsoluteError: 0.0914235934615135 | Loss: 0.0142126699681588 | Epoch: 638 |

MeanAbsoluteError: 0.0964135974645615 | Loss: 0.0152535194434070 | Epoch: 639 |

MeanAbsoluteError: 0.0941481292247772 | Loss: 0.0140724621999349 | Epoch: 640 |

MeanAbsoluteError: 0.0919471979141235 | Loss: 0.0136402633980227 | Epoch: 641 |

MeanAbsoluteError: 0.0897419601678848 | Loss: 0.0128721297246860 | Epoch: 642 |

MeanAbsoluteError: 0.0859987735748291 | Loss: 0.0123512204378979 | Epoch: 643 |

MeanAbsoluteError: 0.0952442437410355 | Loss: 0.0146059012127807 | Epoch: 644 |

MeanAbsoluteError: 0.0882844403386116 | Loss: 0.0131798083862911 | Epoch: 645 |

MeanAbsoluteError: 0.0949488058686256 | Loss: 0.0153795219777385 | Epoch: 646 |

MeanAbsoluteError: 0.0833041220903397 | Loss: 0.0123624810551822 | Epoch: 647 |

MeanAbsoluteError: 0.0986357480287552 | Loss: 0.0167904764682316 | Epoch: 648 |

MeanAbsoluteError: 0.0921415761113167 | Loss: 0.0136074942604137 | Epoch: 649 |

MeanAbsoluteError: 0.0899546146392822 | Loss: 0.0134202059395951 | Epoch: 650 |

MeanAbsoluteError: 0.0902867093682289 | Loss: 0.0139226380417191 | Epoch: 651 |

MeanAbsoluteError: 0.0890051722526550 | Loss: 0.0136951213330576 | Epoch: 652 |

MeanAbsoluteError: 0.0883634686470032 | Loss: 0.0135377957807699 | Epoch: 653 |

MeanAbsoluteError: 0.0934864357113838 | Loss: 0.0142462751980929 | Epoch: 654 |

MeanAbsoluteError: 0.0901531502604485 | Loss: 0.0131909499006845 | Epoch: 655 |

MeanAbsoluteError: 0.0906517803668976 | Loss: 0.0135523621848309 | Epoch: 656 |

MeanAbsoluteError: 0.0944647192955017 | Loss: 0.0147696035142872 | Epoch: 657 |

MeanAbsoluteError: 0.0898716524243355 | Loss: 0.0134419995284892 | Epoch: 658 |

MeanAbsoluteError: 0.0908722653985023 | Loss: 0.0141600228689398 | Epoch: 659 |

MeanAbsoluteError: 0.0914921090006828 | Loss: 0.0136172812132524 | Epoch: 660 |

MeanAbsoluteError: 0.0862129181623459 | Loss: 0.0125896860675615 | Epoch: 661 |

MeanAbsoluteError: 0.0899091660976410 | Loss: 0.0128018571685789 | Epoch: 662 |

MeanAbsoluteError: 0.0886658951640129 | Loss: 0.0123498835242981 | Epoch: 663 |

MeanAbsoluteError: 0.0919820815324783 | Loss: 0.0136060434315247 | Epoch: 664 |

MeanAbsoluteError: 0.0909993052482605 | Loss: 0.0136573489211150 | Epoch: 665 |

MeanAbsoluteError: 0.0857016146183014 | Loss: 0.0125602142562275 | Epoch: 666 |

MeanAbsoluteError: 0.0921309068799019 | Loss: 0.0153196479408251 | Epoch: 667 |

MeanAbsoluteError: 0.0874250382184982 | Loss: 0.0122946061741095 | Epoch: 668 |

MeanAbsoluteError: 0.0904047191143036 | Loss: 0.0137904984501559 | Epoch: 669 |

MeanAbsoluteError: 0.0838717892765999 | Loss: 0.0120195161146694 | Epoch: 670 |

MeanAbsoluteError: 0.0933444276452065 | Loss: 0.0142137601671902 | Epoch: 671 |

MeanAbsoluteError: 0.0826912149786949 | Loss: 0.0116376230075548 | Epoch: 672 |

MeanAbsoluteError: 0.0937707796692848 | Loss: 0.0143063872153895 | Epoch: 673 |

MeanAbsoluteError: 0.0928021073341370 | Loss: 0.0143146147663356 | Epoch: 674 |

MeanAbsoluteError: 0.0910034552216530 | Loss: 0.0140793540118223 | Epoch: 675 |

MeanAbsoluteError: 0.0887886211276054 | Loss: 0.0126948969314374 | Epoch: 676 |

MeanAbsoluteError: 0.0914553999900818 | Loss: 0.0144349705032497 | Epoch: 677 |

MeanAbsoluteError: 0.0864346474409103 | Loss: 0.0122873366463561 | Epoch: 678 |

MeanAbsoluteError: 0.0911565199494362 | Loss: 0.0155455442583479 | Epoch: 679 |

MeanAbsoluteError: 0.0874512195587158 | Loss: 0.0124682045781325 | Epoch: 680 |

MeanAbsoluteError: 0.0868952050805092 | Loss: 0.0122871886139304 | Epoch: 681 |

MeanAbsoluteError: 0.0857532098889351 | Loss: 0.0122076764364950 | Epoch: 682 |

MeanAbsoluteError: 0.0893708765506744 | Loss: 0.0130092634931498 | Epoch: 683 |

MeanAbsoluteError: 0.0933153033256531 | Loss: 0.0140856258919424 | Epoch: 684 |

MeanAbsoluteError: 0.0828882381320000 | Loss: 0.0114508920138144 | Epoch: 685 |

MeanAbsoluteError: 0.0847683623433113 | Loss: 0.0119703660677624 | Epoch: 686 |

MeanAbsoluteError: 0.0841552242636681 | Loss: 0.0119749532328084 | Epoch: 687 |

MeanAbsoluteError: 0.0901032462716103 | Loss: 0.0135006122748988 | Epoch: 688 |

MeanAbsoluteError: 0.0865492448210716 | Loss: 0.0127749186941522 | Epoch: 689 |

MeanAbsoluteError: 0.0901451110839844 | Loss: 0.0138095143953251 | Epoch: 690 |

MeanAbsoluteError: 0.0857677608728409 | Loss: 0.0126865721644572 | Epoch: 691 |

MeanAbsoluteError: 0.0843984186649323 | Loss: 0.0115191478347697 | Epoch: 692 |

MeanAbsoluteError: 0.0862066298723221 | Loss: 0.0126267376492130 | Epoch: 693 |

MeanAbsoluteError: 0.0904658734798431 | Loss: 0.0141702425702715 | Epoch: 694 |

MeanAbsoluteError: 0.0892438367009163 | Loss: 0.0129698373368592 | Epoch: 695 |

MeanAbsoluteError: 0.0888628959655762 | Loss: 0.0137570939340609 | Epoch: 696 |

MeanAbsoluteError: 0.0895384326577187 | Loss: 0.0141815451815637 | Epoch: 697 |

MeanAbsoluteError: 0.0861149206757545 | Loss: 0.0129314811857815 | Epoch: 698 |

MeanAbsoluteError: 0.0846960097551346 | Loss: 0.0120541747261692 | Epoch: 699 |

MeanAbsoluteError: 0.0872166976332664 | Loss: 0.0128992301407076 | Epoch: 700 |

MeanAbsoluteError: 0.0866595730185509 | Loss: 0.0130213049557642 | Epoch: 701 |

MeanAbsoluteError: 0.0869608595967293 | Loss: 0.0124296186906152 | Epoch: 702 |

MeanAbsoluteError: 0.0872699767351151 | Loss: 0.0127087680813565 | Epoch: 703 |

MeanAbsoluteError: 0.0841317698359489 | Loss: 0.0123266009954386 | Epoch: 704 |

MeanAbsoluteError: 0.0838478431105614 | Loss: 0.0118643810501165 | Epoch: 705 |

MeanAbsoluteError: 0.0907689630985260 | Loss: 0.0131419796437694 | Epoch: 706 |

MeanAbsoluteError: 0.0923736691474915 | Loss: 0.0144344044545627 | Epoch: 707 |

MeanAbsoluteError: 0.0874848067760468 | Loss: 0.0127259459533767 | Epoch: 708 |

MeanAbsoluteError: 0.0847158059477806 | Loss: 0.0120570911309843 | Epoch: 709 |

MeanAbsoluteError: 0.0846907198429108 | Loss: 0.0122015894107123 | Epoch: 710 |

MeanAbsoluteError: 0.0923567339777946 | Loss: 0.0139561093621645 | Epoch: 711 |

MeanAbsoluteError: 0.0813598558306694 | Loss: 0.0111752887310467 | Epoch: 712 |

MeanAbsoluteError: 0.0870706215500832 | Loss: 0.0123346273374167 | Epoch: 713 |

MeanAbsoluteError: 0.0867228657007217 | Loss: 0.0130413301346258 | Epoch: 714 |

MeanAbsoluteError: 0.0853150933980942 | Loss: 0.0122990863758908 | Epoch: 715 |

MeanAbsoluteError: 0.0857815966010094 | Loss: 0.0125917645540418 | Epoch: 716 |

MeanAbsoluteError: 0.0787195339798927 | Loss: 0.0110922103355612 | Epoch: 717 |

MeanAbsoluteError: 0.0846261307597160 | Loss: 0.012886540252953 | Epoch: 718 |

MeanAbsoluteError: 0.0919289961457253 | Loss: 0.0144492278527954 | Epoch: 719 |

MeanAbsoluteError: 0.0877352878451347 | Loss: 0.0127414701784922 | Epoch: 720 |

MeanAbsoluteError: 0.0868565812706947 | Loss: 0.0131175352238643 | Epoch: 721 |

MeanAbsoluteError: 0.0793544575572014 | Loss: 0.0110426154763991 | Epoch: 722 |

MeanAbsoluteError: 0.0870934426784515 | Loss: 0.0129277984675718 | Epoch: 723 |

MeanAbsoluteError: 0.0913423523306847 | Loss: 0.0143532960437733 | Epoch: 724 |

MeanAbsoluteError: 0.0847107842564583 | Loss: 0.0122247315116086 | Epoch: 725 |

MeanAbsoluteError: 0.0861899107694626 | Loss: 0.0122133094459423 | Epoch: 726 |

MeanAbsoluteError: 0.0843392908573151 | Loss: 0.0121308150672121 | Epoch: 727 |

MeanAbsoluteError: 0.0903706103563309 | Loss: 0.0136447772903678 | Epoch: 728 |

MeanAbsoluteError: 0.0850528925657272 | Loss: 0.0121053997641866 | Epoch: 729 |

MeanAbsoluteError: 0.0886157676577568 | Loss: 0.0128993192680355 | Epoch: 730 |

MeanAbsoluteError: 0.0836841538548470 | Loss: 0.0119437256065915 | Epoch: 731 |

MeanAbsoluteError: 0.0845359638333321 | Loss: 0.0126823827906888 | Epoch: 732 |

MeanAbsoluteError: 0.0884646475315094 | Loss: 0.0134886265749810 | Epoch: 733 |

MeanAbsoluteError: 0.0781354084610939 | Loss: 0.0104260638561633 | Epoch: 734 |

MeanAbsoluteError: 0.0892178565263748 | Loss: 0.0127959528273641 | Epoch: 735 |

MeanAbsoluteError: 0.0856087207794189 | Loss: 0.0125594279036159 | Epoch: 736 |

MeanAbsoluteError: 0.0869164317846298 | Loss: 0.0136363014497495 | Epoch: 737 |

MeanAbsoluteError: 0.0886043310165405 | Loss: 0.0125537945987404 | Epoch: 738 |

MeanAbsoluteError: 0.0798406973481178 | Loss: 0.0114735874767454 | Epoch: 739 |

MeanAbsoluteError: 0.0847795084118843 | Loss: 0.0123684825277026 | Epoch: 740 |

MeanAbsoluteError: 0.0835280641913414 | Loss: 0.0121070020223851 | Epoch: 741 |

MeanAbsoluteError: 0.0758345872163773 | Loss: 0.0101700595118746 | Epoch: 742 |

MeanAbsoluteError: 0.0866424366831779 | Loss: 0.0130124807901060 | Epoch: 743 |

MeanAbsoluteError: 0.0827291831374168 | Loss: 0.0115952412581343 | Epoch: 744 |

MeanAbsoluteError: 0.0893577411770821 | Loss: 0.0131314118698598 | Epoch: 745 |

MeanAbsoluteError: 0.0815343558788300 | Loss: 0.0107954051935485 | Epoch: 746 |

MeanAbsoluteError: 0.0882723331451416 | Loss: 0.0130692929822302 | Epoch: 747 |

MeanAbsoluteError: 0.0816751047968864 | Loss: 0.0118581207354146 | Epoch: 748 |

MeanAbsoluteError: 0.0859503075480461 | Loss: 0.0119931101474261 | Epoch: 749 |

MeanAbsoluteError: 0.0868570432066917 | Loss: 0.0135347506445639 | Epoch: 750 |

MeanAbsoluteError: 0.0853116810321808 | Loss: 0.0129509823729556 | Epoch: 751 |

MeanAbsoluteError: 0.0842114314436913 | Loss: 0.0114447023558508 | Epoch: 752 |

MeanAbsoluteError: 0.0837864205241203 | Loss: 0.0117758664601449 | Epoch: 753 |

MeanAbsoluteError: 0.0848613902926445 | Loss: 0.0121458923700266 | Epoch: 754 |

MeanAbsoluteError: 0.0860714688897133 | Loss: 0.0132789638465329 | Epoch: 755 |

MeanAbsoluteError: 0.0842629298567772 | Loss: 0.0122338818977657 | Epoch: 756 |

MeanAbsoluteError: 0.0842579752206802 | Loss: 0.0117402255890435 | Epoch: 757 |

MeanAbsoluteError: 0.0839567407965660 | Loss: 0.0113860609992844 | Epoch: 758 |

MeanAbsoluteError: 0.0879777744412422 | Loss: 0.0127438877175397 | Epoch: 759 |

MeanAbsoluteError: 0.0808393731713295 | Loss: 0.0117020922933201 | Epoch: 760 |

MeanAbsoluteError: 0.0866187661886215 | Loss: 0.0123780815792998 | Epoch: 761 |

MeanAbsoluteError: 0.0900759920477867 | Loss: 0.0135301536951738 | Epoch: 762 |

MeanAbsoluteError: 0.0831434130668640 | Loss: 0.0120612022713370 | Epoch: 763 |

MeanAbsoluteError: 0.0854535922408104 | Loss: 0.0122649531954812 | Epoch: 764 |

MeanAbsoluteError: 0.0810860916972160 | Loss: 0.0108526033610906 | Epoch: 765 |

MeanAbsoluteError: 0.0850465297698975 | Loss: 0.0119650142199437 | Epoch: 766 |

MeanAbsoluteError: 0.0840029492974281 | Loss: 0.0119365388162987 | Epoch: 767 |

MeanAbsoluteError: 0.0808563232421875 | Loss: 0.0114324077948064 | Epoch: 768 |

MeanAbsoluteError: 0.0813310593366623 | Loss: 0.0114667138246781 | Epoch: 769 |

MeanAbsoluteError: 0.0834070518612862 | Loss: 0.0116390952894775 | Epoch: 770 |

MeanAbsoluteError: 0.0799439474940300 | Loss: 0.0109760382125023 | Epoch: 771 |

MeanAbsoluteError: 0.0862026885151863 | Loss: 0.0121631451724655 | Epoch: 772 |

MeanAbsoluteError: 0.0773942023515701 | Loss: 0.0100014267676367 | Epoch: 773 |

MeanAbsoluteError: 0.0828531980514526 | Loss: 0.0120185061364706 | Epoch: 774 |

MeanAbsoluteError: 0.0864127799868584 | Loss: 0.0125402316975427 | Epoch: 775 |

MeanAbsoluteError: 0.0896718800067902 | Loss: 0.0133060978676076 | Epoch: 776 |

MeanAbsoluteError: 0.0802150443196297 | Loss: 0.0118395192625515 | Epoch: 777 |

MeanAbsoluteError: 0.0791711807250977 | Loss: 0.0100883760230651 | Epoch: 778 |

MeanAbsoluteError: 0.0781316012144089 | Loss: 0.0107688300175020 | Epoch: 779 |

MeanAbsoluteError: 0.0775944516062737 | Loss: 0.0104368809142549 | Epoch: 780 |

MeanAbsoluteError: 0.0889874026179314 | Loss: 0.0134515109674248 | Epoch: 781 |

MeanAbsoluteError: 0.0862208902835846 | Loss: 0.0122916398667197 | Epoch: 782 |

MeanAbsoluteError: 0.0798108205199242 | Loss: 0.0110968768920672 | Epoch: 783 |

MeanAbsoluteError: 0.0804963931441307 | Loss: 0.0118157476947817 | Epoch: 784 |

MeanAbsoluteError: 0.0776522010564804 | Loss: 0.0103806896301345 | Epoch: 785 |

MeanAbsoluteError: 0.0859545767307281 | Loss: 0.0132408880347975 | Epoch: 786 |

MeanAbsoluteError: 0.0854678750038147 | Loss: 0.0121864469328041 | Epoch: 787 |

MeanAbsoluteError: 0.0822122320532799 | Loss: 0.0114099829130161 | Epoch: 788 |

MeanAbsoluteError: 0.0790107324719429 | Loss: 0.0107910923366580 | Epoch: 789 |

MeanAbsoluteError: 0.0771248340606689 | Loss: 0.0102200264722342 | Epoch: 790 |

MeanAbsoluteError: 0.0901091620326042 | Loss: 0.0141825376562580 | Epoch: 791 |

MeanAbsoluteError: 0.0827606171369553 | Loss: 0.0113155116213602 | Epoch: 792 |

MeanAbsoluteError: 0.0812526568770409 | Loss: 0.0117330015626673 | Epoch: 793 |

MeanAbsoluteError: 0.0786401331424713 | Loss: 0.0105240611347229 | Epoch: 794 |

MeanAbsoluteError: 0.0906210988759995 | Loss: 0.0140195084948148 | Epoch: 795 |

MeanAbsoluteError: 0.0812895819544792 | Loss: 0.0112063214911905 | Epoch: 796 |

MeanAbsoluteError: 0.0839850381016731 | Loss: 0.0120314592663393 | Epoch: 797 |

MeanAbsoluteError: 0.0834412723779678 | Loss: 0.0115786623429934 | Epoch: 798 |

MeanAbsoluteError: 0.0812151804566383 | Loss: 0.0106683049314112 | Epoch: 799 |

MeanAbsoluteError: 0.0763742700219154 | Loss: 0.0103957144342106 | Epoch: 800 |

MeanAbsoluteError: 0.0798877403140068 | Loss: 0.0112059782354754 | Epoch: 801 |

MeanAbsoluteError: 0.0794545784592628 | Loss: 0.0113850245135473 | Epoch: 802 |

MeanAbsoluteError: 0.0882772952318192 | Loss: 0.0136871946473548 | Epoch: 803 |

MeanAbsoluteError: 0.0800916701555252 | Loss: 0.0110358890703962 | Epoch: 804 |

MeanAbsoluteError: 0.0725259333848953 | Loss: 0.0090490159723655 | Epoch: 805 |

MeanAbsoluteError: 0.0803226828575134 | Loss: 0.0112123991530340 | Epoch: 806 |

MeanAbsoluteError: 0.0840378403663635 | Loss: 0.0119105132838498 | Epoch: 807 |

MeanAbsoluteError: 0.0873498767614365 | Loss: 0.0131001907811636 | Epoch: 808 |

MeanAbsoluteError: 0.0818428918719292 | Loss: 0.0120988382856255 | Epoch: 809 |

MeanAbsoluteError: 0.0782985016703606 | Loss: 0.0106757854460254 | Epoch: 810 |

MeanAbsoluteError: 0.0824560895562172 | Loss: 0.0119074617669533 | Epoch: 811 |

MeanAbsoluteError: 0.0821293517947197 | Loss: 0.0119441724711457 | Epoch: 812 |

MeanAbsoluteError: 0.0832219794392586 | Loss: 0.0115171739636571 | Epoch: 813 |

MeanAbsoluteError: 0.0824938267469406 | Loss: 0.0110972798257656 | Epoch: 814 |

MeanAbsoluteError: 0.0840878710150719 | Loss: 0.0117424935035827 | Epoch: 815 |

MeanAbsoluteError: 0.0839302688837051 | Loss: 0.0118463204779709 | Epoch: 816 |

MeanAbsoluteError: 0.0832107365131378 | Loss: 0.0124136097215523 | Epoch: 817 |

MeanAbsoluteError: 0.0769073739647865 | Loss: 0.0101805126779558 | Epoch: 818 |

MeanAbsoluteError: 0.0832949578762054 | Loss: 0.0117550637021971 | Epoch: 819 |

MeanAbsoluteError: 0.0740096569061279 | Loss: 0.0095643966046312 | Epoch: 820 |

MeanAbsoluteError: 0.0754168927669525 | Loss: 0.0096059564733393 | Epoch: 821 |

MeanAbsoluteError: 0.0801299884915352 | Loss: 0.0109681259503122 | Epoch: 822 |

MeanAbsoluteError: 0.0877290070056915 | Loss: 0.0132411294273576 | Epoch: 823 |

MeanAbsoluteError: 0.0738099738955498 | Loss: 0.0091930555650833 | Epoch: 824 |

MeanAbsoluteError: 0.0823216438293457 | Loss: 0.0113966476301236 | Epoch: 825 |

MeanAbsoluteError: 0.0730762258172035 | Loss: 0.0096490480711994 | Epoch: 826 |

MeanAbsoluteError: 0.0775209143757820 | Loss: 0.0108585520722651 | Epoch: 827 |

MeanAbsoluteError: 0.0800625681877136 | Loss: 0.0106971635242614 | Epoch: 828 |

MeanAbsoluteError: 0.0830442681908607 | Loss: 0.0114131947926944 | Epoch: 829 |

MeanAbsoluteError: 0.0812834650278091 | Loss: 0.0124094398109749 | Epoch: 830 |

MeanAbsoluteError: 0.0802404880523682 | Loss: 0.0111840228424990 | Epoch: 831 |

MeanAbsoluteError: 0.0865611061453819 | Loss: 0.0122170231806558 | Epoch: 832 |

MeanAbsoluteError: 0.0766393169760704 | Loss: 0.0107930416374681 | Epoch: 833 |

MeanAbsoluteError: 0.0792775452136993 | Loss: 0.0110985868957990 | Epoch: 834 |

MeanAbsoluteError: 0.0718979611992836 | Loss: 0.0087367886219439 | Epoch: 835 |

MeanAbsoluteError: 0.0837861150503159 | Loss: 0.0112630593621968 | Epoch: 836 |

MeanAbsoluteError: 0.0794481039047241 | Loss: 0.0112295329577501 | Epoch: 837 |

MeanAbsoluteError: 0.0799805447459221 | Loss: 0.0108191633181499 | Epoch: 838 |

MeanAbsoluteError: 0.0753138363361359 | Loss: 0.0100799758657983 | Epoch: 839 |

MeanAbsoluteError: 0.0814361795783043 | Loss: 0.0113643640919690 | Epoch: 840 |

MeanAbsoluteError: 0.0781028494238853 | Loss: 0.0107340678073494 | Epoch: 841 |

MeanAbsoluteError: 0.0771855488419533 | Loss: 0.0102697229348511 | Epoch: 842 |

MeanAbsoluteError: 0.0809403955936432 | Loss: 0.0111956146650179 | Epoch: 843 |

MeanAbsoluteError: 0.0764201506972313 | Loss: 0.0105490690810499 | Epoch: 844 |

MeanAbsoluteError: 0.0738255828619003 | Loss: 0.0097503840384767 | Epoch: 845 |

MeanAbsoluteError: 0.0785881280899048 | Loss: 0.0117377337430662 | Epoch: 846 |

MeanAbsoluteError: 0.0779688134789467 | Loss: 0.0101995708775818 | Epoch: 847 |

MeanAbsoluteError: 0.0807364657521248 | Loss: 0.0107182800451119 | Epoch: 848 |

MeanAbsoluteError: 0.0804117023944855 | Loss: 0.0113706592589006 | Epoch: 849 |

MeanAbsoluteError: 0.0828294157981873 | Loss: 0.0110238781937854 | Epoch: 850 |

MeanAbsoluteError: 0.0774060785770416 | Loss: 0.0108154441271048 | Epoch: 851 |

MeanAbsoluteError: 0.0771163329482079 | Loss: 0.0102891369386793 | Epoch: 852 |

MeanAbsoluteError: 0.0783403366804123 | Loss: 0.0100651222461117 | Epoch: 853 |

MeanAbsoluteError: 0.0766426697373390 | Loss: 0.0103630085376911 | Epoch: 854 |

MeanAbsoluteError: 0.0791056975722313 | Loss: 0.0108683706768594 | Epoch: 855 |

MeanAbsoluteError: 0.0764939263463020 | Loss: 0.0102093208984782 | Epoch: 856 |

MeanAbsoluteError: 0.0814418941736221 | Loss: 0.0116528933390024 | Epoch: 857 |

MeanAbsoluteError: 0.0757196247577667 | Loss: 0.0098749243262379 | Epoch: 858 |

MeanAbsoluteError: 0.0742248371243477 | Loss: 0.0095135539109712 | Epoch: 859 |

MeanAbsoluteError: 0.0841125324368477 | Loss: 0.0119662338884882 | Epoch: 860 |

MeanAbsoluteError: 0.0771385431289673 | Loss: 0.0106761798496397 | Epoch: 861 |

MeanAbsoluteError: 0.0801075771450996 | Loss: 0.0110092316208102 | Epoch: 862 |

MeanAbsoluteError: 0.0781409740447998 | Loss: 0.0107365397114578 | Epoch: 863 |

MeanAbsoluteError: 0.0781718343496323 | Loss: 0.0104807782462619 | Epoch: 864 |

MeanAbsoluteError: 0.0794635787606239 | Loss: 0.0108283075337143 | Epoch: 865 |

MeanAbsoluteError: 0.0785385444760323 | Loss: 0.0109409504162371 | Epoch: 866 |

MeanAbsoluteError: 0.0750286206603050 | Loss: 0.0095710122664362 | Epoch: 867 |

MeanAbsoluteError: 0.0773174315690994 | Loss: 0.0098690391287285 | Epoch: 868 |

MeanAbsoluteError: 0.0767803639173508 | Loss: 0.0110145149399553 | Epoch: 869 |

MeanAbsoluteError: 0.0764931514859200 | Loss: 0.0102086972078018 | Epoch: 870 |

MeanAbsoluteError: 0.0775896161794662 | Loss: 0.0095922199140356 | Epoch: 871 |

MeanAbsoluteError: 0.0743670612573624 | Loss: 0.0099460433783421 | Epoch: 872 |

MeanAbsoluteError: 0.0775201916694641 | Loss: 0.0103023139870250 | Epoch: 873 |

MeanAbsoluteError: 0.0769089236855507 | Loss: 0.0104139984305948 | Epoch: 874 |

MeanAbsoluteError: 0.0778351575136185 | Loss: 0.0103990351249134 | Epoch: 875 |

MeanAbsoluteError: 0.0746545642614365 | Loss: 0.0097688974583192 | Epoch: 876 |

MeanAbsoluteError: 0.0754441544413567 | Loss: 0.0104262638159222 | Epoch: 877 |

MeanAbsoluteError: 0.0789302289485931 | Loss: 0.0107850675039663 | Epoch: 878 |

MeanAbsoluteError: 0.0779957920312881 | Loss: 0.0104319902939460 | Epoch: 879 |

MeanAbsoluteError: 0.0791888460516930 | Loss: 0.0109203641870196 | Epoch: 880 |

MeanAbsoluteError: 0.0734420269727707 | Loss: 0.0102968009971179 | Epoch: 881 |

MeanAbsoluteError: 0.0805623084306717 | Loss: 0.0116654501038829 | Epoch: 882 |

MeanAbsoluteError: 0.0795516967773438 | Loss: 0.0110667387986662 | Epoch: 883 |

MeanAbsoluteError: 0.0825117006897926 | Loss: 0.0111786892855404 | Epoch: 884 |

MeanAbsoluteError: 0.0732105821371078 | Loss: 0.0095361146165912 | Epoch: 885 |

MeanAbsoluteError: 0.0807303711771965 | Loss: 0.0109143830911247 | Epoch: 886 |

MeanAbsoluteError: 0.0835117027163506 | Loss: 0.0127478042116854 | Epoch: 887 |

MeanAbsoluteError: 0.0746336653828621 | Loss: 0.0095123651331717 | Epoch: 888 |

MeanAbsoluteError: 0.0762807503342628 | Loss: 0.0101715381572285 | Epoch: 889 |

MeanAbsoluteError: 0.0816012024879456 | Loss: 0.0111705045261381 | Epoch: 890 |

MeanAbsoluteError: 0.0728234797716141 | Loss: 0.0093935145442568 | Epoch: 891 |

MeanAbsoluteError: 0.0753166228532791 | Loss: 0.0100770461460039 | Epoch: 892 |

MeanAbsoluteError: 0.0754622742533684 | Loss: 0.0094174520477827 | Epoch: 893 |

MeanAbsoluteError: 0.0743034332990646 | Loss: 0.0098858077853705 | Epoch: 894 |

MeanAbsoluteError: 0.0756712555885315 | Loss: 0.0095969021834026 | Epoch: 895 |

MeanAbsoluteError: 0.0754574611783028 | Loss: 0.0102671565598575 | Epoch: 896 |

MeanAbsoluteError: 0.0775607079267502 | Loss: 0.0100980512174040 | Epoch: 897 |

MeanAbsoluteError: 0.0826487243175507 | Loss: 0.0122165740561710 | Epoch: 898 |

MeanAbsoluteError: 0.0699848905205727 | Loss: 0.0087210551842266 | Epoch: 899 |

MeanAbsoluteError: 0.0771350860595703 | Loss: 0.0101687256007184 | Epoch: 900 |

MeanAbsoluteError: 0.0748658180236816 | Loss: 0.0101067419190076 | Epoch: 901 |

MeanAbsoluteError: 0.0805472210049629 | Loss: 0.0112878356254078 | Epoch: 902 |

MeanAbsoluteError: 0.0746419876813889 | Loss: 0.0098066572048750 | Epoch: 903 |

MeanAbsoluteError: 0.0746952071785927 | Loss: 0.0098825468095553 | Epoch: 904 |

MeanAbsoluteError: 0.0787869393825531 | Loss: 0.0108093548449203 | Epoch: 905 |

MeanAbsoluteError: 0.0752108991146088 | Loss: 0.0097216395382808 | Epoch: 906 |

MeanAbsoluteError: 0.0848613604903221 | Loss: 0.0121929631560245 | Epoch: 907 |

MeanAbsoluteError: 0.0791209191083908 | Loss: 0.0117048701097398 | Epoch: 908 |

MeanAbsoluteError: 0.0808530300855637 | Loss: 0.0113955594588333 | Epoch: 909 |

MeanAbsoluteError: 0.0770772770047188 | Loss: 0.0103067371755454 | Epoch: 910 |

MeanAbsoluteError: 0.0764493718743324 | Loss: 0.0102760676636361 | Epoch: 911 |

MeanAbsoluteError: 0.0741509124636650 | Loss: 0.0092806246098189 | Epoch: 912 |

MeanAbsoluteError: 0.0816884860396385 | Loss: 0.0119303727195802 | Epoch: 913 |

MeanAbsoluteError: 0.0733480826020241 | Loss: 0.0095463680885344 | Epoch: 914 |

MeanAbsoluteError: 0.0761298462748528 | Loss: 0.0099002483793690 | Epoch: 915 |

MeanAbsoluteError: 0.0747096389532089 | Loss: 0.0095522906991149 | Epoch: 916 |

MeanAbsoluteError: 0.0766460001468658 | Loss: 0.0108985542458746 | Epoch: 917 |

MeanAbsoluteError: 0.0786527916789055 | Loss: 0.0104521911375923 | Epoch: 918 |

MeanAbsoluteError: 0.0804752409458160 | Loss: 0.0116180918461760 | Epoch: 919 |

MeanAbsoluteError: 0.0777545571327209 | Loss: 0.0103211307371870 | Epoch: 920 |

MeanAbsoluteError: 0.0779756531119347 | Loss: 0.0102706307186115 | Epoch: 921 |

MeanAbsoluteError: 0.0779359862208366 | Loss: 0.0108993076187714 | Epoch: 922 |

MeanAbsoluteError: 0.0759358704090118 | Loss: 0.0103246881165008 | Epoch: 923 |

MeanAbsoluteError: 0.0763984173536301 | Loss: 0.0100957613766999 | Epoch: 924 |

MeanAbsoluteError: 0.0703286752104759 | Loss: 0.0090990765301346 | Epoch: 925 |

MeanAbsoluteError: 0.0735248699784279 | Loss: 0.0098544619079621 | Epoch: 926 |

MeanAbsoluteError: 0.0746264457702637 | Loss: 0.0100789326563730 | Epoch: 927 |

MeanAbsoluteError: 0.0730664730072021 | Loss: 0.0090286806012470 | Epoch: 928 |

MeanAbsoluteError: 0.0797500163316727 | Loss: 0.0102156727723680 | Epoch: 929 |

MeanAbsoluteError: 0.0764885991811752 | Loss: 0.0098379029554053 | Epoch: 930 |

MeanAbsoluteError: 0.0715815722942352 | Loss: 0.0089564489267408 | Epoch: 931 |

MeanAbsoluteError: 0.0794423967599869 | Loss: 0.0112672232636032 | Epoch: 932 |

MeanAbsoluteError: 0.0724268928170204 | Loss: 0.0090378722118112 | Epoch: 933 |

MeanAbsoluteError: 0.0754097327589989 | Loss: 0.0099079379982322 | Epoch: 934 |

MeanAbsoluteError: 0.0757404565811157 | Loss: 0.0097645578289424 | Epoch: 935 |

MeanAbsoluteError: 0.0724296793341637 | Loss: 0.0089836852836985 | Epoch: 936 |

MeanAbsoluteError: 0.0719136893749237 | Loss: 0.0088832404560041 | Epoch: 937 |

MeanAbsoluteError: 0.0790272280573845 | Loss: 0.0099801414067042 | Epoch: 938 |

MeanAbsoluteError: 0.0677152350544930 | Loss: 0.0081527328717190 | Epoch: 939 |

MeanAbsoluteError: 0.0721540227532387 | Loss: 0.0091816935042637 | Epoch: 940 |

MeanAbsoluteError: 0.0765955522656441 | Loss: 0.0094809635374744 | Epoch: 941 |

MeanAbsoluteError: 0.0727129876613617 | Loss: 0.0095080480841231 | Epoch: 942 |

MeanAbsoluteError: 0.0773292258381844 | Loss: 0.0103785896316579 | Epoch: 943 |

MeanAbsoluteError: 0.0792438313364983 | Loss: 0.0110065474185346 | Epoch: 944 |

MeanAbsoluteError: 0.0739024132490158 | Loss: 0.0095757057363274 | Epoch: 945 |

MeanAbsoluteError: 0.0727039873600006 | Loss: 0.0089056415118163 | Epoch: 946 |

MeanAbsoluteError: 0.0793486312031746 | Loss: 0.0110480698788888 | Epoch: 947 |

MeanAbsoluteError: 0.0771549344062805 | Loss: 0.0105916457230827 | Epoch: 948 |

MeanAbsoluteError: 0.0703706443309784 | Loss: 0.0090822723888171 | Epoch: 949 |

MeanAbsoluteError: 0.0740337669849396 | Loss: 0.0102533181999267 | Epoch: 950 |

MeanAbsoluteError: 0.0765396952629089 | Loss: 0.0100489563544882 | Epoch: 951 |

MeanAbsoluteError: 0.0765357688069344 | Loss: 0.0101444794183408 | Epoch: 952 |

MeanAbsoluteError: 0.0734773129224777 | Loss: 0.0099917671943695 | Epoch: 953 |

MeanAbsoluteError: 0.0715711712837219 | Loss: 0.0095782629013411 | Epoch: 954 |

MeanAbsoluteError: 0.0729345530271530 | Loss: 0.0094018803842967 | Epoch: 955 |

MeanAbsoluteError: 0.0733283907175064 | Loss: 0.0096717089774756 | Epoch: 956 |

MeanAbsoluteError: 0.0664042308926582 | Loss: 0.0075158126316516 | Epoch: 957 |

MeanAbsoluteError: 0.0849085971713066 | Loss: 0.0119881409450318 | Epoch: 958 |

MeanAbsoluteError: 0.0715690329670906 | Loss: 0.0091539401113626 | Epoch: 959 |

MeanAbsoluteError: 0.0777297839522362 | Loss: 0.0104211996043159 | Epoch: 960 |

MeanAbsoluteError: 0.0760367810726166 | Loss: 0.0104746409421205 | Epoch: 961 |

MeanAbsoluteError: 0.0776511132717133 | Loss: 0.0111037619700316 | Epoch: 962 |

MeanAbsoluteError: 0.0707922950387001 | Loss: 0.0090281871824603 | Epoch: 963 |

MeanAbsoluteError: 0.0769897550344467 | Loss: 0.0107690826061783 | Epoch: 964 |

MeanAbsoluteError: 0.0697699263691902 | Loss: 0.0081867781741797 | Epoch: 965 |

MeanAbsoluteError: 0.0764012634754181 | Loss: 0.0108604474635892 | Epoch: 966 |

MeanAbsoluteError: 0.0792110189795494 | Loss: 0.0113623218954369 | Epoch: 967 |

MeanAbsoluteError: 0.0680814906954765 | Loss: 0.0083186678276009 | Epoch: 968 |

MeanAbsoluteError: 0.0725850313901901 | Loss: 0.0094845377973009 | Epoch: 969 |

MeanAbsoluteError: 0.0762547701597214 | Loss: 0.0102596534014204 | Epoch: 970 |

MeanAbsoluteError: 0.0696619078516960 | Loss: 0.0081080010741910 | Epoch: 971 |

MeanAbsoluteError: 0.0711411610245705 | Loss: 0.0090589344280428 | Epoch: 972 |

MeanAbsoluteError: 0.0727351084351540 | Loss: 0.0092486638832391 | Epoch: 973 |

MeanAbsoluteError: 0.0737860798835754 | Loss: 0.0092965269653359 | Epoch: 974 |

MeanAbsoluteError: 0.0753273069858551 | Loss: 0.0099358842343402 | Epoch: 975 |

MeanAbsoluteError: 0.0756720975041389 | Loss: 0.0098584534610563 | Epoch: 976 |

MeanAbsoluteError: 0.0716525614261627 | Loss: 0.0090443917666077 | Epoch: 977 |

MeanAbsoluteError: 0.0730086863040924 | Loss: 0.0097757532730611 | Epoch: 978 |

MeanAbsoluteError: 0.0764009654521942 | Loss: 0.0101318568076385 | Epoch: 979 |

MeanAbsoluteError: 0.0712708234786987 | Loss: 0.0090927594126212 | Epoch: 980 |

MeanAbsoluteError: 0.0709680244326591 | Loss: 0.0096254755349582 | Epoch: 981 |

MeanAbsoluteError: 0.0715130865573883 | Loss: 0.0096350274606569 | Epoch: 982 |

MeanAbsoluteError: 0.0728397071361542 | Loss: 0.0099155692999648 | Epoch: 983 |

MeanAbsoluteError: 0.0771391838788986 | Loss: 0.0107261698419461 | Epoch: 984 |

MeanAbsoluteError: 0.0724724307656288 | Loss: 0.0090657099912035 | Epoch: 985 |

MeanAbsoluteError: 0.0712971761822701 | Loss: 0.0090209147415165 | Epoch: 986 |

MeanAbsoluteError: 0.0698899328708649 | Loss: 0.0085721351750302 | Epoch: 987 |

MeanAbsoluteError: 0.0693163201212883 | Loss: 0.0077545857433385 | Epoch: 988 |

MeanAbsoluteError: 0.0778072550892830 | Loss: 0.0107944230326878 | Epoch: 989 |

MeanAbsoluteError: 0.0761123374104500 | Loss: 0.0094622577789490 | Epoch: 990 |

MeanAbsoluteError: 0.0749316513538361 | Loss: 0.0101282367263229 | Epoch: 991 |

MeanAbsoluteError: 0.0717362016439438 | Loss: 0.0088800959480674 | Epoch: 992 |

MeanAbsoluteError: 0.0655228495597839 | Loss: 0.0076068037412733 | Epoch: 993 |

MeanAbsoluteError: 0.0738191530108452 | Loss: 0.0094339414276571 | Epoch: 994 |

MeanAbsoluteError: 0.0797284319996834 | Loss: 0.0114746082314766 | Epoch: 995 |

MeanAbsoluteError: 0.0736414566636086 | Loss: 0.0100833321006697 | Epoch: 996 |

MeanAbsoluteError: 0.0766720548272133 | Loss: 0.0103790204787462 | Epoch: 997 |

MeanAbsoluteError: 0.0690694302320480 | Loss: 0.0089957126947896 | Epoch: 998 |

MeanAbsoluteError: 0.0725758820772171 | Loss: 0.0092693577688866 | Epoch: 999 |

MeanAbsoluteError: 0.0757641568779945 | Loss: 0.0099406880475484 | Epoch: 1000 |

MeanAbsoluteError: 0.0723375603556633 | Loss: 0.0101366889583248 | Epoch: 1001 |

MeanAbsoluteError: 0.0713929086923599 | Loss: 0.0091776353289364 | Epoch: 1002 |

MeanAbsoluteError: 0.0694112330675125 | Loss: 0.0086696119510937 | Epoch: 1003 |

MeanAbsoluteError: 0.0732303783297539 | Loss: 0.0095423594975849 | Epoch: 1004 |

MeanAbsoluteError: 0.0719602331519127 | Loss: 0.0082926946849329 | Epoch: 1005 |

MeanAbsoluteError: 0.0707664713263512 | Loss: 0.0089822108592731 | Epoch: 1006 |

MeanAbsoluteError: 0.0708745792508125 | Loss: 0.0093132891812517 | Epoch: 1007 |

MeanAbsoluteError: 0.0725696086883545 | Loss: 0.0094351426910725 | Epoch: 1008 |

MeanAbsoluteError: 0.0679365545511246 | Loss: 0.0079000075886628 | Epoch: 1009 |

MeanAbsoluteError: 0.0667665898799896 | Loss: 0.0092165257916046 | Epoch: 1010 |

MeanAbsoluteError: 0.0719863474369049 | Loss: 0.0090799885893405 | Epoch: 1011 |

MeanAbsoluteError: 0.0761761292815208 | Loss: 0.0102922801974637 | Epoch: 1012 |

MeanAbsoluteError: 0.0675073340535164 | Loss: 0.0081174084200272 | Epoch: 1013 |

MeanAbsoluteError: 0.0678270235657692 | Loss: 0.0079732331828806 | Epoch: 1014 |

MeanAbsoluteError: 0.0752263218164444 | Loss: 0.0106562207843914 | Epoch: 1015 |

MeanAbsoluteError: 0.0706823244690895 | Loss: 0.0092785234641300 | Epoch: 1016 |

MeanAbsoluteError: 0.0702643245458603 | Loss: 0.0087310705716845 | Epoch: 1017 |

MeanAbsoluteError: 0.0732280239462852 | Loss: 0.0096235771773142 | Epoch: 1018 |

MeanAbsoluteError: 0.0692768841981888 | Loss: 0.0082831174120171 | Epoch: 1019 |

MeanAbsoluteError: 0.0753551498055458 | Loss: 0.0096873664389326 | Epoch: 1020 |

MeanAbsoluteError: 0.0788179561495781 | Loss: 0.0112903491220156 | Epoch: 1021 |

MeanAbsoluteError: 0.0696059018373489 | Loss: 0.0085636919625783 | Epoch: 1022 |

MeanAbsoluteError: 0.0750660821795464 | Loss: 0.0098088718126261 | Epoch: 1023 |

MeanAbsoluteError: 0.0688222870230675 | Loss: 0.0089297082419822 | Epoch: 1024 |

MeanAbsoluteError: 0.0734671279788017 | Loss: 0.0099654179563125 | Epoch: 1025 |

MeanAbsoluteError: 0.0677940100431442 | Loss: 0.0084685865843433 | Epoch: 1026 |

MeanAbsoluteError: 0.0730460286140442 | Loss: 0.0097838113074319 | Epoch: 1027 |

MeanAbsoluteError: 0.0737103074789047 | Loss: 0.0096964498836314 | Epoch: 1028 |

MeanAbsoluteError: 0.0693713650107384 | Loss: 0.0093506347872608 | Epoch: 1029 |

MeanAbsoluteError: 0.0701157450675964 | Loss: 0.0087472067823546 | Epoch: 1030 |

MeanAbsoluteError: 0.0719802454113960 | Loss: 0.0096490722182110 | Epoch: 1031 |

MeanAbsoluteError: 0.0744744092226028 | Loss: 0.0104563620192145 | Epoch: 1032 |

MeanAbsoluteError: 0.0747015625238419 | Loss: 0.0103342440272293 | Epoch: 1033 |

MeanAbsoluteError: 0.0731337293982506 | Loss: 0.0096799701658892 | Epoch: 1034 |

MeanAbsoluteError: 0.0718859583139420 | Loss: 0.0092115516729730 | Epoch: 1035 |

MeanAbsoluteError: 0.0719785094261169 | Loss: 0.0090696068745698 | Epoch: 1036 |

MeanAbsoluteError: 0.0762908458709717 | Loss: 0.0108155495210970 | Epoch: 1037 |

MeanAbsoluteError: 0.0699102133512497 | Loss: 0.0090078560039175 | Epoch: 1038 |

MeanAbsoluteError: 0.0694839209318161 | Loss: 0.0086378032860375 | Epoch: 1039 |

MeanAbsoluteError: 0.0731324180960655 | Loss: 0.0096406716015917 | Epoch: 1040 |

MeanAbsoluteError: 0.0663361102342606 | Loss: 0.0076710213937743 | Epoch: 1041 |

MeanAbsoluteError: 0.0674593001604080 | Loss: 0.0081189934132271 | Epoch: 1042 |

MeanAbsoluteError: 0.0701092034578323 | Loss: 0.0089317029489708 | Epoch: 1043 |

MeanAbsoluteError: 0.0735797211527824 | Loss: 0.0096080499193583 | Epoch: 1044 |

MeanAbsoluteError: 0.0681488066911697 | Loss: 0.0079945703036507 | Epoch: 1045 |

MeanAbsoluteError: 0.0694965645670891 | Loss: 0.0086281360519691 | Epoch: 1046 |

MeanAbsoluteError: 0.0708992630243301 | Loss: 0.0085548651159055 | Epoch: 1047 |

MeanAbsoluteError: 0.0678436905145645 | Loss: 0.0080009132574439 | Epoch: 1048 |

MeanAbsoluteError: 0.0731303021311760 | Loss: 0.0092584842301646 | Epoch: 1049 |

MeanAbsoluteError: 0.0756253078579903 | Loss: 0.0105017330964862 | Epoch: 1050 |

MeanAbsoluteError: 0.0721068158745766 | Loss: 0.0089157650391523 | Epoch: 1051 |

MeanAbsoluteError: 0.0698562487959862 | Loss: 0.0081887228436244 | Epoch: 1052 |

MeanAbsoluteError: 0.0688799545168877 | Loss: 0.0084516765458996 | Epoch: 1053 |

MeanAbsoluteError: 0.0791916921734810 | Loss: 0.0116683767591409 | Epoch: 1054 |

MeanAbsoluteError: 0.0730694830417633 | Loss: 0.0094450911650529 | Epoch: 1055 |

MeanAbsoluteError: 0.0685431510210037 | Loss: 0.0087991150578576 | Epoch: 1056 |

MeanAbsoluteError: 0.0678988918662071 | Loss: 0.0082361182645339 | Epoch: 1057 |

MeanAbsoluteError: 0.0718125626444817 | Loss: 0.0095836025548518 | Epoch: 1058 |

MeanAbsoluteError: 0.0689691454172134 | Loss: 0.0083297175485738 | Epoch: 1059 |

MeanAbsoluteError: 0.0660716146230698 | Loss: 0.0076677496425570 | Epoch: 1060 |

MeanAbsoluteError: 0.0669567584991455 | Loss: 0.0078201846442971 | Epoch: 1061 |

MeanAbsoluteError: 0.0718378126621246 | Loss: 0.0092942472408079 | Epoch: 1062 |

MeanAbsoluteError: 0.0683273673057556 | Loss: 0.0085367330375690 | Epoch: 1063 |

MeanAbsoluteError: 0.0689010992646217 | Loss: 0.0087301500673228 | Epoch: 1064 |

MeanAbsoluteError: 0.0643932297825813 | Loss: 0.0071408245173613 | Epoch: 1065 |

MeanAbsoluteError: 0.0712036862969398 | Loss: 0.0091494820010121 | Epoch: 1066 |

MeanAbsoluteError: 0.0682067498564720 | Loss: 0.0087686679004764 | Epoch: 1067 |

MeanAbsoluteError: 0.0710598751902580 | Loss: 0.0090669604683111 | Epoch: 1068 |

MeanAbsoluteError: 0.0675958842039108 | Loss: 0.0081680967913417 | Epoch: 1069 |

MeanAbsoluteError: 0.0707667469978333 | Loss: 0.0094727789251677 | Epoch: 1070 |

MeanAbsoluteError: 0.0663241595029831 | Loss: 0.0078773448887538 | Epoch: 1071 |

MeanAbsoluteError: 0.0724717304110527 | Loss: 0.0093914666551670 | Epoch: 1072 |

MeanAbsoluteError: 0.0717313811182976 | Loss: 0.0090600793244569 | Epoch: 1073 |

MeanAbsoluteError: 0.0719677880406380 | Loss: 0.0091367837757692 | Epoch: 1074 |

MeanAbsoluteError: 0.0674020349979401 | Loss: 0.0085003050091230 | Epoch: 1075 |

MeanAbsoluteError: 0.0694231316447258 | Loss: 0.0089321442251821 | Epoch: 1076 |

MeanAbsoluteError: 0.0661793649196625 | Loss: 0.0070797218290924 | Epoch: 1077 |

MeanAbsoluteError: 0.0693086609244347 | Loss: 0.0086002728496048 | Epoch: 1078 |

MeanAbsoluteError: 0.0682182610034943 | Loss: 0.0084075029252078 | Epoch: 1079 |

MeanAbsoluteError: 0.0723419412970543 | Loss: 0.0090570476208813 | Epoch: 1080 |

MeanAbsoluteError: 0.0651640966534615 | Loss: 0.0080746152600235 | Epoch: 1081 |

MeanAbsoluteError: 0.0617406852543354 | Loss: 0.0069656755405716 | Epoch: 1082 |

MeanAbsoluteError: 0.0633678138256073 | Loss: 0.0072673743660562 | Epoch: 1083 |

MeanAbsoluteError: 0.0673019960522652 | Loss: 0.0079992112914139 | Epoch: 1084 |

MeanAbsoluteError: 0.0694029703736305 | Loss: 0.0083690348218806 | Epoch: 1085 |

MeanAbsoluteError: 0.0638776794075966 | Loss: 0.0074368117840095 | Epoch: 1086 |

MeanAbsoluteError: 0.0665798112750053 | Loss: 0.0080212725897824 | Epoch: 1087 |

MeanAbsoluteError: 0.0722908452153206 | Loss: 0.0093754962139550 | Epoch: 1088 |

MeanAbsoluteError: 0.0764328837394714 | Loss: 0.0104183597763282 | Epoch: 1089 |

MeanAbsoluteError: 0.0698969215154648 | Loss: 0.0095780186166182 | Epoch: 1090 |

MeanAbsoluteError: 0.0676443576812744 | Loss: 0.0079124876828185 | Epoch: 1091 |

MeanAbsoluteError: 0.0707011148333549 | Loss: 0.0085442703529285 | Epoch: 1092 |

MeanAbsoluteError: 0.0677080750465393 | Loss: 0.0080793829660009 | Epoch: 1093 |

MeanAbsoluteError: 0.0691639408469200 | Loss: 0.0086508790500860 | Epoch: 1094 |

MeanAbsoluteError: 0.0643378347158432 | Loss: 0.0072232539331981 | Epoch: 1095 |

MeanAbsoluteError: 0.0713246688246727 | Loss: 0.0092748370420062 | Epoch: 1096 |

MeanAbsoluteError: 0.0711861550807953 | Loss: 0.0087414858707052 | Epoch: 1097 |

MeanAbsoluteError: 0.0624565519392490 | Loss: 0.0068722203207653 | Epoch: 1098 |

MeanAbsoluteError: 0.0719688534736633 | Loss: 0.0088836418749270 | Epoch: 1099 |

MeanAbsoluteError: 0.0663241520524025 | Loss: 0.0074788226276473 | Epoch: 1100 |

MeanAbsoluteError: 0.0681140944361687 | Loss: 0.0081330509797651 | Epoch: 1101 |

MeanAbsoluteError: 0.0636119991540909 | Loss: 0.0069956652772574 | Epoch: 1102 |

MeanAbsoluteError: 0.0701044127345085 | Loss: 0.0090232997812564 | Epoch: 1103 |

MeanAbsoluteError: 0.0704035684466362 | Loss: 0.0088492787976065 | Epoch: 1104 |

MeanAbsoluteError: 0.0680999532341957 | Loss: 0.0082811595315676 | Epoch: 1105 |

MeanAbsoluteError: 0.0747594609856606 | Loss: 0.0104180746887384 | Epoch: 1106 |

MeanAbsoluteError: 0.0672458261251450 | Loss: 0.0086416085725553 | Epoch: 1107 |

MeanAbsoluteError: 0.0632396042346954 | Loss: 0.0071163129012992 | Epoch: 1108 |

MeanAbsoluteError: 0.0686077326536179 | Loss: 0.0078957221242308 | Epoch: 1109 |

MeanAbsoluteError: 0.0693401023745537 | Loss: 0.0088114524478200 | Epoch: 1110 |

MeanAbsoluteError: 0.0736144557595253 | Loss: 0.0092911955611695 | Epoch: 1111 |

MeanAbsoluteError: 0.0670956149697304 | Loss: 0.0083384482148055 | Epoch: 1112 |

MeanAbsoluteError: 0.0610097646713257 | Loss: 0.0068200649288337 | Epoch: 1113 |

MeanAbsoluteError: 0.0702242180705070 | Loss: 0.0086200210745604 | Epoch: 1114 |

MeanAbsoluteError: 0.0643501579761505 | Loss: 0.0074991043059951 | Epoch: 1115 |

MeanAbsoluteError: 0.0699807405471802 | Loss: 0.0084590349494465 | Epoch: 1116 |

MeanAbsoluteError: 0.0649750903248787 | Loss: 0.0079310609535363 | Epoch: 1117 |

MeanAbsoluteError: 0.0683524161577225 | Loss: 0.0086109045504539 | Epoch: 1118 |

MeanAbsoluteError: 0.0705199390649796 | Loss: 0.0088848530270419 | Epoch: 1119 |

MeanAbsoluteError: 0.0647038966417313 | Loss: 0.0078909225898678 | Epoch: 1120 |

MeanAbsoluteError: 0.0658162757754326 | Loss: 0.0074180137240910 | Epoch: 1121 |

MeanAbsoluteError: 0.0630656927824020 | Loss: 0.0073879817066942 | Epoch: 1122 |

MeanAbsoluteError: 0.0704290568828583 | Loss: 0.0097256720506872 | Epoch: 1123 |

MeanAbsoluteError: 0.0673821568489075 | Loss: 0.0086733655763722 | Epoch: 1124 |

MeanAbsoluteError: 0.0633160769939423 | Loss: 0.0074544591983105 | Epoch: 1125 |

MeanAbsoluteError: 0.0698489397764206 | Loss: 0.0092363261471716 | Epoch: 1126 |

MeanAbsoluteError: 0.0667150542140007 | Loss: 0.0085544286544367 | Epoch: 1127 |

MeanAbsoluteError: 0.0622264780104160 | Loss: 0.0073222268723475 | Epoch: 1128 |

MeanAbsoluteError: 0.0700443759560585 | Loss: 0.0087137780142560 | Epoch: 1129 |

MeanAbsoluteError: 0.0636391043663025 | Loss: 0.0080010952279796 | Epoch: 1130 |

MeanAbsoluteError: 0.0685411989688873 | Loss: 0.0085516337645155 | Epoch: 1131 |

MeanAbsoluteError: 0.0694776251912117 | Loss: 0.0088909361958455 | Epoch: 1132 |

MeanAbsoluteError: 0.0710195824503899 | Loss: 0.0093507448901801 | Epoch: 1133 |

MeanAbsoluteError: 0.0692139938473701 | Loss: 0.0095294034390342 | Epoch: 1134 |

MeanAbsoluteError: 0.0640364289283752 | Loss: 0.0074145980762357 | Epoch: 1135 |

MeanAbsoluteError: 0.0730346888303757 | Loss: 0.0099281933723008 | Epoch: 1136 |

MeanAbsoluteError: 0.0645968988537788 | Loss: 0.0076082818453475 | Epoch: 1137 |

MeanAbsoluteError: 0.0681149587035179 | Loss: 0.0081982624193188 | Epoch: 1138 |

MeanAbsoluteError: 0.0568541400134563 | Loss: 0.0061345131113436 | Epoch: 1139 |

MeanAbsoluteError: 0.0721034705638885 | Loss: 0.0091914032175312 | Epoch: 1140 |

MeanAbsoluteError: 0.0628835782408714 | Loss: 0.0072235697410118 | Epoch: 1141 |

MeanAbsoluteError: 0.0682444572448730 | Loss: 0.0088440893554556 | Epoch: 1142 |

MeanAbsoluteError: 0.0657172799110413 | Loss: 0.0081542717427102 | Epoch: 1143 |

MeanAbsoluteError: 0.0689317211508751 | Loss: 0.0088074087502294 | Epoch: 1144 |

MeanAbsoluteError: 0.0665861964225769 | Loss: 0.0082033753929803 | Epoch: 1145 |

MeanAbsoluteError: 0.0685911178588867 | Loss: 0.0083404243922996 | Epoch: 1146 |

MeanAbsoluteError: 0.0693357139825821 | Loss: 0.0087487636282337 | Epoch: 1147 |

MeanAbsoluteError: 0.0690763443708420 | Loss: 0.0085716901036115 | Epoch: 1148 |

MeanAbsoluteError: 0.0654565021395683 | Loss: 0.0075826229208421 | Epoch: 1149 |

MeanAbsoluteError: 0.0637311413884163 | Loss: 0.0080189988608921 | Epoch: 1150 |

MeanAbsoluteError: 0.0630521327257156 | Loss: 0.0073405471866863 | Epoch: 1151 |

MeanAbsoluteError: 0.0673010796308517 | Loss: 0.0082781683623974 | Epoch: 1152 |

MeanAbsoluteError: 0.0652497187256813 | Loss: 0.0075149046512282 | Epoch: 1153 |

MeanAbsoluteError: 0.0644122660160065 | Loss: 0.0076896131125432 | Epoch: 1154 |

MeanAbsoluteError: 0.0695091784000397 | Loss: 0.0089211251551933 | Epoch: 1155 |

MeanAbsoluteError: 0.0651471838355064 | Loss: 0.0078405609992236 | Epoch: 1156 |

MeanAbsoluteError: 0.0650300532579422 | Loss: 0.0082767158206358 | Epoch: 1157 |

MeanAbsoluteError: 0.0638088062405586 | Loss: 0.0075246719357043 | Epoch: 1158 |

MeanAbsoluteError: 0.0634868741035461 | Loss: 0.0084015505952266 | Epoch: 1159 |

MeanAbsoluteError: 0.0657244622707367 | Loss: 0.0075654348954837 | Epoch: 1160 |

MeanAbsoluteError: 0.0686446800827980 | Loss: 0.0085990752160918 | Epoch: 1161 |

MeanAbsoluteError: 0.0593461729586124 | Loss: 0.0063627362946863 | Epoch: 1162 |

MeanAbsoluteError: 0.0659839585423470 | Loss: 0.0084608737313435 | Epoch: 1163 |

MeanAbsoluteError: 0.0680202767252922 | Loss: 0.0081164759048746 | Epoch: 1164 |

MeanAbsoluteError: 0.0614863261580467 | Loss: 0.0067693859304488 | Epoch: 1165 |

MeanAbsoluteError: 0.0639742016792297 | Loss: 0.0075848067779407 | Epoch: 1166 |

MeanAbsoluteError: 0.0661838501691818 | Loss: 0.0072266113433094 | Epoch: 1167 |

MeanAbsoluteError: 0.0694292783737183 | Loss: 0.0087912789970475 | Epoch: 1168 |

MeanAbsoluteError: 0.0628383010625839 | Loss: 0.0072622589636194 | Epoch: 1169 |

MeanAbsoluteError: 0.0737675130367279 | Loss: 0.0095739141812858 | Epoch: 1170 |

MeanAbsoluteError: 0.0631414502859116 | Loss: 0.0071509282972784 | Epoch: 1171 |

MeanAbsoluteError: 0.0655021741986275 | Loss: 0.0075843964378995 | Epoch: 1172 |

MeanAbsoluteError: 0.0635970532894135 | Loss: 0.0075995598383588 | Epoch: 1173 |

MeanAbsoluteError: 0.0690304338932037 | Loss: 0.0088975602488790 | Epoch: 1174 |

MeanAbsoluteError: 0.0650118961930275 | Loss: 0.0073082571775012 | Epoch: 1175 |

MeanAbsoluteError: 0.0636664181947708 | Loss: 0.0084230540087447 | Epoch: 1176 |

MeanAbsoluteError: 0.0656239017844200 | Loss: 0.0078917141198508 | Epoch: 1177 |

MeanAbsoluteError: 0.0640999376773834 | Loss: 0.0077136761686173 | Epoch: 1178 |

MeanAbsoluteError: 0.0677836909890175 | Loss: 0.0082124785569370 | Epoch: 1179 |

MeanAbsoluteError: 0.0680667757987976 | Loss: 0.0084519529765930 | Epoch: 1180 |

MeanAbsoluteError: 0.0644348189234734 | Loss: 0.0077755609569188 | Epoch: 1181 |

MeanAbsoluteError: 0.0679789781570435 | Loss: 0.0082269871740209 | Epoch: 1182 |

MeanAbsoluteError: 0.0693719983100891 | Loss: 0.0088834195684467 | Epoch: 1183 |

MeanAbsoluteError: 0.0708234310150146 | Loss: 0.0084363166937449 | Epoch: 1184 |

MeanAbsoluteError: 0.0643603578209877 | Loss: 0.0071625918989594 | Epoch: 1185 |

MeanAbsoluteError: 0.0663974136114120 | Loss: 0.0078951712282408 | Epoch: 1186 |

MeanAbsoluteError: 0.0646793991327286 | Loss: 0.0071780992643107 | Epoch: 1187 |

MeanAbsoluteError: 0.0636620670557022 | Loss: 0.0079789436916326 | Epoch: 1188 |

MeanAbsoluteError: 0.0629383102059364 | Loss: 0.0070229293719361 | Epoch: 1189 |

MeanAbsoluteError: 0.0677320435643196 | Loss: 0.0090679464868541 | Epoch: 1190 |

MeanAbsoluteError: 0.0631995424628258 | Loss: 0.0069566987120197 | Epoch: 1191 |

MeanAbsoluteError: 0.0657962933182716 | Loss: 0.0077235924074193 | Epoch: 1192 |

MeanAbsoluteError: 0.0663131698966026 | Loss: 0.0092412873707660 | Epoch: 1193 |

MeanAbsoluteError: 0.0706317797303200 | Loss: 0.0083772922731199 | Epoch: 1194 |

MeanAbsoluteError: 0.0617574639618397 | Loss: 0.0068849398712094 | Epoch: 1195 |

MeanAbsoluteError: 0.0634381845593452 | Loss: 0.0072911292174892 | Epoch: 1196 |

MeanAbsoluteError: 0.0610828660428524 | Loss: 0.0069242807231785 | Epoch: 1197 |

MeanAbsoluteError: 0.0656199231743813 | Loss: 0.0077633254823741 | Epoch: 1198 |

MeanAbsoluteError: 0.0640247687697411 | Loss: 0.0071229057681436 | Epoch: 1199 |

MeanAbsoluteError: 0.0605994984507561 | Loss: 0.0064563069170739 | Epoch: 1200 |

MeanAbsoluteError: 0.0638386979699135 | Loss: 0.0079527828085217 | Epoch: 1201 |

MeanAbsoluteError: 0.0638684779405594 | Loss: 0.0074921829904876 | Epoch: 1202 |

MeanAbsoluteError: 0.0665121003985405 | Loss: 0.0076121065080103 | Epoch: 1203 |

MeanAbsoluteError: 0.0627066642045975 | Loss: 0.0067927504340817 | Epoch: 1204 |

MeanAbsoluteError: 0.0616116859018803 | Loss: 0.0066306355282965 | Epoch: 1205 |

MeanAbsoluteError: 0.0623222216963768 | Loss: 0.0068787825482286 | Epoch: 1206 |

MeanAbsoluteError: 0.0655873268842697 | Loss: 0.0077651797419336 | Epoch: 1207 |

MeanAbsoluteError: 0.0612384974956512 | Loss: 0.0075284201707594 | Epoch: 1208 |

MeanAbsoluteError: 0.0662722215056419 | Loss: 0.0078261838824255 | Epoch: 1209 |

MeanAbsoluteError: 0.0682407394051552 | Loss: 0.0078729740777150 | Epoch: 1210 |

MeanAbsoluteError: 0.0614073686301708 | Loss: 0.0067319621486240 | Epoch: 1211 |

MeanAbsoluteError: 0.0699118971824646 | Loss: 0.0092297816770345 | Epoch: 1212 |

MeanAbsoluteError: 0.0663583725690842 | Loss: 0.0077028569242248 | Epoch: 1213 |

MeanAbsoluteError: 0.0629780292510986 | Loss: 0.0072989186171132 | Epoch: 1214 |

MeanAbsoluteError: 0.0623923316597939 | Loss: 0.0071688902984048 | Epoch: 1215 |

MeanAbsoluteError: 0.0588259436190128 | Loss: 0.0064029444973736 | Epoch: 1216 |

MeanAbsoluteError: 0.0604680627584457 | Loss: 0.0068228776528940 | Epoch: 1217 |

MeanAbsoluteError: 0.0586263462901115 | Loss: 0.0064769137612226 | Epoch: 1218 |

MeanAbsoluteError: 0.0602800883352757 | Loss: 0.0070878212951963 | Epoch: 1219 |

MeanAbsoluteError: 0.0674337223172188 | Loss: 0.0092628806611174 | Epoch: 1220 |

MeanAbsoluteError: 0.0636176913976669 | Loss: 0.0072799733443208 | Epoch: 1221 |

MeanAbsoluteError: 0.0637678205966949 | Loss: 0.0073069029384836 | Epoch: 1222 |

MeanAbsoluteError: 0.0659564435482025 | Loss: 0.0083267193833550 | Epoch: 1223 |

MeanAbsoluteError: 0.0627924129366875 | Loss: 0.0073029461163727 | Epoch: 1224 |

MeanAbsoluteError: 0.0610148310661316 | Loss: 0.0069422768032139 | Epoch: 1225 |

MeanAbsoluteError: 0.0654839575290680 | Loss: 0.0077286712363032 | Epoch: 1226 |

MeanAbsoluteError: 0.0612850524485111 | Loss: 0.0070277456571178 | Epoch: 1227 |

MeanAbsoluteError: 0.0624229349195957 | Loss: 0.0072544638851347 | Epoch: 1228 |

MeanAbsoluteError: 0.0678682103753090 | Loss: 0.0082256965307442 | Epoch: 1229 |

MeanAbsoluteError: 0.0637050569057465 | Loss: 0.0075593736559676 | Epoch: 1230 |

MeanAbsoluteError: 0.0608091801404953 | Loss: 0.0070597851579108 | Epoch: 1231 |

MeanAbsoluteError: 0.0629947930574417 | Loss: 0.0079606243237504 | Epoch: 1232 |

MeanAbsoluteError: 0.0592392273247242 | Loss: 0.0063577746347922 | Epoch: 1233 |

MeanAbsoluteError: 0.0641319230198860 | Loss: 0.0074856198683119 | Epoch: 1234 |

MeanAbsoluteError: 0.0640417709946632 | Loss: 0.0076579014409557 | Epoch: 1235 |

MeanAbsoluteError: 0.0648472234606743 | Loss: 0.0072228103398447 | Epoch: 1236 |

MeanAbsoluteError: 0.0654282867908478 | Loss: 0.0078260383226977 | Epoch: 1237 |

MeanAbsoluteError: 0.0594246536493301 | Loss: 0.0063910981766579 | Epoch: 1238 |

MeanAbsoluteError: 0.0600025355815887 | Loss: 0.0071964854974552 | Epoch: 1239 |

MeanAbsoluteError: 0.0626812949776649 | Loss: 0.0070920031542361 | Epoch: 1240 |

MeanAbsoluteError: 0.0636621788144112 | Loss: 0.0071963769529702 | Epoch: 1241 |

MeanAbsoluteError: 0.0603998377919197 | Loss: 0.0066154810394194 | Epoch: 1242 |

MeanAbsoluteError: 0.0580286346375942 | Loss: 0.0068103034042815 | Epoch: 1243 |

MeanAbsoluteError: 0.0658731386065483 | Loss: 0.0077163204426567 | Epoch: 1244 |

MeanAbsoluteError: 0.0682158097624779 | Loss: 0.0090915094003140 | Epoch: 1245 |

MeanAbsoluteError: 0.0638194233179092 | Loss: 0.0076256748243153 | Epoch: 1246 |

MeanAbsoluteError: 0.0638536140322685 | Loss: 0.0074687880871534 | Epoch: 1247 |

MeanAbsoluteError: 0.0676750391721725 | Loss: 0.0083274144616007 | Epoch: 1248 |

MeanAbsoluteError: 0.0632640346884727 | Loss: 0.0076300418087703 | Epoch: 1249 |

MeanAbsoluteError: 0.0602971278131008 | Loss: 0.0067283271338359 | Epoch: 1250 |

MeanAbsoluteError: 0.0630327090620995 | Loss: 0.0074337795241854 | Epoch: 1251 |

MeanAbsoluteError: 0.0637849867343903 | Loss: 0.0076189291305491 | Epoch: 1252 |

MeanAbsoluteError: 0.0609539151191711 | Loss: 0.0072467654468710 | Epoch: 1253 |

MeanAbsoluteError: 0.0717900246381760 | Loss: 0.0093050736988031 | Epoch: 1254 |

MeanAbsoluteError: 0.0611215718090534 | Loss: 0.0069163748118808 | Epoch: 1255 |

MeanAbsoluteError: 0.0634086951613426 | Loss: 0.0077655467807199 | Epoch: 1256 |

MeanAbsoluteError: 0.0623031929135323 | Loss: 0.0069076488720990 | Epoch: 1257 |

MeanAbsoluteError: 0.0590523481369019 | Loss: 0.0064767875665954 | Epoch: 1258 |

MeanAbsoluteError: 0.0597368627786636 | Loss: 0.0058900471955470 | Epoch: 1259 |

MeanAbsoluteError: 0.0659222677350044 | Loss: 0.0078090839892623 | Epoch: 1260 |

MeanAbsoluteError: 0.0587507560849190 | Loss: 0.0060456933487876 | Epoch: 1261 |

MeanAbsoluteError: 0.0604133717715740 | Loss: 0.0064466639775977 | Epoch: 1262 |

MeanAbsoluteError: 0.0602659657597542 | Loss: 0.0066147537159486 | Epoch: 1263 |

MeanAbsoluteError: 0.0659210681915283 | Loss: 0.0075018033431297 | Epoch: 1264 |

MeanAbsoluteError: 0.0629273429512978 | Loss: 0.0077414131077482 | Epoch: 1265 |

MeanAbsoluteError: 0.0602748617529869 | Loss: 0.0069400191610233 | Epoch: 1266 |

MeanAbsoluteError: 0.0618423782289028 | Loss: 0.0073407786262032 | Epoch: 1267 |

MeanAbsoluteError: 0.0601481124758720 | Loss: 0.0072917424881962 | Epoch: 1268 |

MeanAbsoluteError: 0.0637676566839218 | Loss: 0.0080641012059035 | Epoch: 1269 |

MeanAbsoluteError: 0.0600695498287678 | Loss: 0.0066601895076504 | Epoch: 1270 |

MeanAbsoluteError: 0.0643502548336983 | Loss: 0.0085518484201020 | Epoch: 1271 |

MeanAbsoluteError: 0.0621754974126816 | Loss: 0.0069625924370969 | Epoch: 1272 |

MeanAbsoluteError: 0.0601709112524986 | Loss: 0.0060629628669267 | Epoch: 1273 |

MeanAbsoluteError: 0.0641725510358810 | Loss: 0.0077398539259896 | Epoch: 1274 |

MeanAbsoluteError: 0.0588267147541046 | Loss: 0.0070124945870217 | Epoch: 1275 |

MeanAbsoluteError: 0.0626857057213783 | Loss: 0.0071745190692188 | Epoch: 1276 |

MeanAbsoluteError: 0.0624087601900101 | Loss: 0.0067718264816601 | Epoch: 1277 |

MeanAbsoluteError: 0.0588923916220665 | Loss: 0.0065521929023392 | Epoch: 1278 |

MeanAbsoluteError: 0.0618937946856022 | Loss: 0.0068137911330511 | Epoch: 1279 |

MeanAbsoluteError: 0.0579067021608353 | Loss: 0.0057953725017069 | Epoch: 1280 |

MeanAbsoluteError: 0.0647194832563400 | Loss: 0.0076527326156490 | Epoch: 1281 |

MeanAbsoluteError: 0.0623487271368504 | Loss: 0.0070749801584073 | Epoch: 1282 |

MeanAbsoluteError: 0.0650510042905807 | Loss: 0.0080797809162201 | Epoch: 1283 |

MeanAbsoluteError: 0.0673635900020599 | Loss: 0.0075853963613433 | Epoch: 1284 |

MeanAbsoluteError: 0.0673955306410789 | Loss: 0.0084427857683174 | Epoch: 1285 |

MeanAbsoluteError: 0.0636771395802498 | Loss: 0.0071116609923289 | Epoch: 1286 |

MeanAbsoluteError: 0.0557297952473164 | Loss: 0.0058030620970627 | Epoch: 1287 |

MeanAbsoluteError: 0.0586284138262272 | Loss: 0.0068149602077998 | Epoch: 1288 |

MeanAbsoluteError: 0.0616724006831646 | Loss: 0.0071897057637761 | Epoch: 1289 |

MeanAbsoluteError: 0.0647143125534058 | Loss: 0.0073602350142028 | Epoch: 1290 |

MeanAbsoluteError: 0.0561532340943813 | Loss: 0.0060248755173719 | Epoch: 1291 |

MeanAbsoluteError: 0.0588872097432613 | Loss: 0.0066914881002716 | Epoch: 1292 |

MeanAbsoluteError: 0.0592726506292820 | Loss: 0.0062342988254629 | Epoch: 1293 |

MeanAbsoluteError: 0.0595637112855911 | Loss: 0.0066505825210455 | Epoch: 1294 |

MeanAbsoluteError: 0.0549105443060398 | Loss: 0.0056616590865148 | Epoch: 1295 |

MeanAbsoluteError: 0.0625222697854042 | Loss: 0.0068665233757565 | Epoch: 1296 |

MeanAbsoluteError: 0.0566158629953861 | Loss: 0.0062104222044521 | Epoch: 1297 |

MeanAbsoluteError: 0.0618584044277668 | Loss: 0.0070241265785050 | Epoch: 1298 |

MeanAbsoluteError: 0.0594248883426189 | Loss: 0.0069459147612118 | Epoch: 1299 |

MeanAbsoluteError: 0.0622272603213787 | Loss: 0.0074573160023526 | Epoch: 1300 |

MeanAbsoluteError: 0.0594698637723923 | Loss: 0.0069741047012303 | Epoch: 1301 |

MeanAbsoluteError: 0.0579961128532887 | Loss: 0.0069740403233360 | Epoch: 1302 |

MeanAbsoluteError: 0.0599656663835049 | Loss: 0.0072865175659535 | Epoch: 1303 |

MeanAbsoluteError: 0.0624936595559120 | Loss: 0.0073876735518328 | Epoch: 1304 |

MeanAbsoluteError: 0.0610631369054317 | Loss: 0.0067687466252695 | Epoch: 1305 |

MeanAbsoluteError: 0.0625288039445877 | Loss: 0.0072661710742106 | Epoch: 1306 |

MeanAbsoluteError: 0.0603088513016701 | Loss: 0.0070992012679199 | Epoch: 1307 |

MeanAbsoluteError: 0.0615134350955486 | Loss: 0.0068495385882003 | Epoch: 1308 |

MeanAbsoluteError: 0.0575127936899662 | Loss: 0.0061127125966262 | Epoch: 1309 |

MeanAbsoluteError: 0.0672807991504669 | Loss: 0.0086072244690392 | Epoch: 1310 |

MeanAbsoluteError: 0.0644313320517540 | Loss: 0.0084025245387420 | Epoch: 1311 |

MeanAbsoluteError: 0.0646878629922867 | Loss: 0.0071357543653236 | Epoch: 1312 |

MeanAbsoluteError: 0.0562869086861610 | Loss: 0.0058477657335849 | Epoch: 1313 |

MeanAbsoluteError: 0.0620988830924034 | Loss: 0.0074806166989447 | Epoch: 1314 |

MeanAbsoluteError: 0.0671301335096359 | Loss: 0.0084829986490452 | Epoch: 1315 |

MeanAbsoluteError: 0.0634774938225746 | Loss: 0.0082823823812214 | Epoch: 1316 |

MeanAbsoluteError: 0.0584264993667603 | Loss: 0.0060556682418413 | Epoch: 1317 |

MeanAbsoluteError: 0.0553562752902508 | Loss: 0.0055762075724730 | Epoch: 1318 |

MeanAbsoluteError: 0.0631681606173515 | Loss: 0.0072884432606467 | Epoch: 1319 |

MeanAbsoluteError: 0.0585636831820011 | Loss: 0.0067937828821899 | Epoch: 1320 |

MeanAbsoluteError: 0.0601073652505875 | Loss: 0.0069069218236837 | Epoch: 1321 |

MeanAbsoluteError: 0.0592857040464878 | Loss: 0.0060135061820135 | Epoch: 1322 |

MeanAbsoluteError: 0.0585643053054810 | Loss: 0.0065385152722350 | Epoch: 1323 |

MeanAbsoluteError: 0.0556948855519295 | Loss: 0.0055800235328085 | Epoch: 1324 |

MeanAbsoluteError: 0.0630860254168510 | Loss: 0.0076132753168349 | Epoch: 1325 |

MeanAbsoluteError: 0.0604993179440498 | Loss: 0.0065574741142821 | Epoch: 1326 |

MeanAbsoluteError: 0.0557281114161015 | Loss: 0.0057680278202558 | Epoch: 1327 |

MeanAbsoluteError: 0.0650261491537094 | Loss: 0.0077713587357236 | Epoch: 1328 |

MeanAbsoluteError: 0.0594450384378433 | Loss: 0.0072241933765842 | Epoch: 1329 |

MeanAbsoluteError: 0.0586771965026855 | Loss: 0.0064483105690548 | Epoch: 1330 |

MeanAbsoluteError: 0.0587195083498955 | Loss: 0.0074206920600288 | Epoch: 1331 |

MeanAbsoluteError: 0.0637977644801140 | Loss: 0.0067886768792232 | Epoch: 1332 |

MeanAbsoluteError: 0.0652621686458588 | Loss: 0.0074189749639481 | Epoch: 1333 |

MeanAbsoluteError: 0.0586406141519547 | Loss: 0.0060554788097761 | Epoch: 1334 |

MeanAbsoluteError: 0.0655524730682373 | Loss: 0.0084028575081902 | Epoch: 1335 |

MeanAbsoluteError: 0.0602464750409126 | Loss: 0.0068814275343660 | Epoch: 1336 |

MeanAbsoluteError: 0.0621792227029800 | Loss: 0.0076427560026059 | Epoch: 1337 |

MeanAbsoluteError: 0.0623363032937050 | Loss: 0.0070257983475676 | Epoch: 1338 |

MeanAbsoluteError: 0.0573655441403389 | Loss: 0.0054982766404767 | Epoch: 1339 |

MeanAbsoluteError: 0.0578860044479370 | Loss: 0.0061039369699392 | Epoch: 1340 |

MeanAbsoluteError: 0.0609294176101685 | Loss: 0.0068390948525121 | Epoch: 1341 |

MeanAbsoluteError: 0.0595331117510796 | Loss: 0.0063120169434114 | Epoch: 1342 |

MeanAbsoluteError: 0.0590211860835552 | Loss: 0.0062593080263468 | Epoch: 1343 |

MeanAbsoluteError: 0.0525709092617035 | Loss: 0.0055392229639582 | Epoch: 1344 |

MeanAbsoluteError: 0.0588330067694187 | Loss: 0.0065999297063536 | Epoch: 1345 |

MeanAbsoluteError: 0.0631908699870110 | Loss: 0.0074271243907181 | Epoch: 1346 |

MeanAbsoluteError: 0.0613631233572960 | Loss: 0.0074646979543468 | Epoch: 1347 |

MeanAbsoluteError: 0.0590438134968281 | Loss: 0.0071434897894263 | Epoch: 1348 |

MeanAbsoluteError: 0.0591630823910236 | Loss: 0.0071320187094777 | Epoch: 1349 |

MeanAbsoluteError: 0.0613437257707119 | Loss: 0.0066808529830450 | Epoch: 1350 |

MeanAbsoluteError: 0.0550273023545742 | Loss: 0.0059833803693377 | Epoch: 1351 |

MeanAbsoluteError: 0.0572957471013069 | Loss: 0.0063646000387477 | Epoch: 1352 |

MeanAbsoluteError: 0.0646844133734703 | Loss: 0.0076713488257034 | Epoch: 1353 |

MeanAbsoluteError: 0.0533348023891449 | Loss: 0.0051347972557172 | Epoch: 1354 |

MeanAbsoluteError: 0.0610402487218380 | Loss: 0.0073597275284070 | Epoch: 1355 |

MeanAbsoluteError: 0.0563038513064384 | Loss: 0.0062525593141618 | Epoch: 1356 |

MeanAbsoluteError: 0.0623960494995117 | Loss: 0.0076754165287033 | Epoch: 1357 |

MeanAbsoluteError: 0.0579704865813255 | Loss: 0.0062934564885048 | Epoch: 1358 |

MeanAbsoluteError: 0.0580323673784733 | Loss: 0.0066176986267116 | Epoch: 1359 |

MeanAbsoluteError: 0.0603172034025192 | Loss: 0.0070527732018794 | Epoch: 1360 |

MeanAbsoluteError: 0.0595178268849850 | Loss: 0.0068137397707809 | Epoch: 1361 |

MeanAbsoluteError: 0.0578610934317112 | Loss: 0.0060816108078507 | Epoch: 1362 |

MeanAbsoluteError: 0.0583778880536556 | Loss: 0.0067611638918364 | Epoch: 1363 |

MeanAbsoluteError: 0.0596525371074677 | Loss: 0.0068535130712644 | Epoch: 1364 |

MeanAbsoluteError: 0.0559104867279530 | Loss: 0.0058111705333674 | Epoch: 1365 |

MeanAbsoluteError: 0.0601880252361298 | Loss: 0.0059277274716806 | Epoch: 1366 |

MeanAbsoluteError: 0.0602724775671959 | Loss: 0.0067496870191826 | Epoch: 1367 |

MeanAbsoluteError: 0.0616057515144348 | Loss: 0.0070057805495283 | Epoch: 1368 |

MeanAbsoluteError: 0.0639656782150269 | Loss: 0.0074389386884123 | Epoch: 1369 |

MeanAbsoluteError: 0.0590878799557686 | Loss: 0.0075507379388485 | Epoch: 1370 |

MeanAbsoluteError: 0.0593984089791775 | Loss: 0.0066270784619250 | Epoch: 1371 |

MeanAbsoluteError: 0.0600770190358162 | Loss: 0.0073270322577506 | Epoch: 1372 |

MeanAbsoluteError: 0.0575873292982578 | Loss: 0.0060493746074402 | Epoch: 1373 |

MeanAbsoluteError: 0.0611264593899250 | Loss: 0.0066616562008373 | Epoch: 1374 |

MeanAbsoluteError: 0.0570595860481262 | Loss: 0.0061177108495031 | Epoch: 1375 |

MeanAbsoluteError: 0.0590573251247406 | Loss: 0.0069289187466105 | Epoch: 1376 |

MeanAbsoluteError: 0.0594652816653252 | Loss: 0.0067402777333435 | Epoch: 1377 |

MeanAbsoluteError: 0.0568242780864239 | Loss: 0.0059960014817868 | Epoch: 1378 |

MeanAbsoluteError: 0.0597791299223900 | Loss: 0.0068522613890188 | Epoch: 1379 |

MeanAbsoluteError: 0.0582293048501015 | Loss: 0.0064122937783501 | Epoch: 1380 |

MeanAbsoluteError: 0.0601521506905556 | Loss: 0.0064453515933807 | Epoch: 1381 |

MeanAbsoluteError: 0.0561903454363346 | Loss: 0.0060621916408551 | Epoch: 1382 |

MeanAbsoluteError: 0.0584788769483566 | Loss: 0.0064061350609518 | Epoch: 1383 |

MeanAbsoluteError: 0.0577516108751297 | Loss: 0.0066056292869689 | Epoch: 1384 |

MeanAbsoluteError: 0.0612967237830162 | Loss: 0.0067308303606114 | Epoch: 1385 |

MeanAbsoluteError: 0.0579466819763184 | Loss: 0.0063529075566233 | Epoch: 1386 |

MeanAbsoluteError: 0.0639848709106445 | Loss: 0.0070912594742307 | Epoch: 1387 |

MeanAbsoluteError: 0.0587789267301559 | Loss: 0.0062040600233498 | Epoch: 1388 |

MeanAbsoluteError: 0.0607269592583179 | Loss: 0.0067732133742902 | Epoch: 1389 |

MeanAbsoluteError: 0.0613276921212673 | Loss: 0.0070115517686645 | Epoch: 1390 |

MeanAbsoluteError: 0.0588996559381485 | Loss: 0.0068311178821993 | Epoch: 1391 |

MeanAbsoluteError: 0.0577615797519684 | Loss: 0.0061380676289749 | Epoch: 1392 |

MeanAbsoluteError: 0.0582601614296436 | Loss: 0.0064320448323269 | Epoch: 1393 |

MeanAbsoluteError: 0.0658429414033890 | Loss: 0.0076372725332112 | Epoch: 1394 |

MeanAbsoluteError: 0.0565693043172359 | Loss: 0.0058412261591820 | Epoch: 1395 |

MeanAbsoluteError: 0.0569558329880238 | Loss: 0.0064749789136295 | Epoch: 1396 |

MeanAbsoluteError: 0.0563307143747807 | Loss: 0.0060115725524095 | Epoch: 1397 |

MeanAbsoluteError: 0.0563693158328533 | Loss: 0.0061697309390790 | Epoch: 1398 |

MeanAbsoluteError: 0.0614904612302780 | Loss: 0.0064544608261713 | Epoch: 1399 |

MeanAbsoluteError: 0.0548842884600163 | Loss: 0.0055805883152667 | Epoch: 1400 |

MeanAbsoluteError: 0.0557563863694668 | Loss: 0.0056326612958886 | Epoch: 1401 |

MeanAbsoluteError: 0.0624559335410595 | Loss: 0.0067359105225235 | Epoch: 1402 |

MeanAbsoluteError: 0.0552742183208466 | Loss: 0.0062173305620248 | Epoch: 1403 |

MeanAbsoluteError: 0.0623117238283157 | Loss: 0.0073411542934628 | Epoch: 1404 |

MeanAbsoluteError: 0.0573842041194439 | Loss: 0.0057818155378482 | Epoch: 1405 |

MeanAbsoluteError: 0.0559544488787651 | Loss: 0.0059102018125607 | Epoch: 1406 |

MeanAbsoluteError: 0.0587244294583797 | Loss: 0.0065422553435443 | Epoch: 1407 |

MeanAbsoluteError: 0.0600770823657513 | Loss: 0.0059424061224126 | Epoch: 1408 |

MeanAbsoluteError: 0.0608702786266804 | Loss: 0.0072758367889643 | Epoch: 1409 |

MeanAbsoluteError: 0.0586856380105019 | Loss: 0.0066306086830203 | Epoch: 1410 |

MeanAbsoluteError: 0.0574161447584629 | Loss: 0.0060549706985694 | Epoch: 1411 |

MeanAbsoluteError: 0.0583676956593990 | Loss: 0.0063128329048292 | Epoch: 1412 |

MeanAbsoluteError: 0.0566170886158943 | Loss: 0.0059740128747217 | Epoch: 1413 |

MeanAbsoluteError: 0.0528923086822033 | Loss: 0.0055720553557073 | Epoch: 1414 |

MeanAbsoluteError: 0.0579725652933121 | Loss: 0.0065579826983837 | Epoch: 1415 |

MeanAbsoluteError: 0.0557838045060635 | Loss: 0.0060128550286451 | Epoch: 1416 |

MeanAbsoluteError: 0.0618123114109039 | Loss: 0.0078098540362771 | Epoch: 1417 |

MeanAbsoluteError: 0.0574797429144382 | Loss: 0.0060527243328337 | Epoch: 1418 |

MeanAbsoluteError: 0.0591564439237118 | Loss: 0.0069078163054292 | Epoch: 1419 |

MeanAbsoluteError: 0.0563187673687935 | Loss: 0.0067976737817662 | Epoch: 1420 |

MeanAbsoluteError: 0.0591088868677616 | Loss: 0.0065378680860946 | Epoch: 1421 |

MeanAbsoluteError: 0.0564135685563087 | Loss: 0.0059805855708692 | Epoch: 1422 |

MeanAbsoluteError: 0.0571763738989830 | Loss: 0.0061837889390396 | Epoch: 1423 |

MeanAbsoluteError: 0.0582145899534225 | Loss: 0.0061869906513675 | Epoch: 1424 |

MeanAbsoluteError: 0.0524145551025867 | Loss: 0.0057837775284133 | Epoch: 1425 |

MeanAbsoluteError: 0.0559653416275978 | Loss: 0.0063784418028844 | Epoch: 1426 |

MeanAbsoluteError: 0.0564562417566776 | Loss: 0.0062507339088734 | Epoch: 1427 |

MeanAbsoluteError: 0.0562570504844189 | Loss: 0.0060353516241594 | Epoch: 1428 |

MeanAbsoluteError: 0.0584495477378368 | Loss: 0.0066283909364514 | Epoch: 1429 |

MeanAbsoluteError: 0.0598415993154049 | Loss: 0.0064357079134182 | Epoch: 1430 |

MeanAbsoluteError: 0.0590827576816082 | Loss: 0.0069418430483832 | Epoch: 1431 |

MeanAbsoluteError: 0.0519671812653542 | Loss: 0.0053607662739159 | Epoch: 1432 |

MeanAbsoluteError: 0.0585345886647701 | Loss: 0.0069909224350765 | Epoch: 1433 |

MeanAbsoluteError: 0.0623789578676224 | Loss: 0.0078136000643159 | Epoch: 1434 |

MeanAbsoluteError: 0.0573706887662411 | Loss: 0.0063272417403156 | Epoch: 1435 |

MeanAbsoluteError: 0.0552980490028858 | Loss: 0.0054508830875178 | Epoch: 1436 |

MeanAbsoluteError: 0.0565602742135525 | Loss: 0.0059347634245205 | Epoch: 1437 |

MeanAbsoluteError: 0.0601707585155964 | Loss: 0.0070364983431076 | Epoch: 1438 |

MeanAbsoluteError: 0.0586043857038021 | Loss: 0.0066380368627870 | Epoch: 1439 |

MeanAbsoluteError: 0.0557993873953819 | Loss: 0.0063275659259822 | Epoch: 1440 |

MeanAbsoluteError: 0.0584771856665611 | Loss: 0.0061750227016455 | Epoch: 1441 |

MeanAbsoluteError: 0.0558768026530743 | Loss: 0.0058388078034629 | Epoch: 1442 |

MeanAbsoluteError: 0.0602018088102341 | Loss: 0.0065983752051397 | Epoch: 1443 |

MeanAbsoluteError: 0.0604710839688778 | Loss: 0.0068428725943947 | Epoch: 1444 |

MeanAbsoluteError: 0.0585154667496681 | Loss: 0.0068361961480211 | Epoch: 1445 |

MeanAbsoluteError: 0.0586277954280376 | Loss: 0.0064402937030294 | Epoch: 1446 |

MeanAbsoluteError: 0.0596406571567059 | Loss: 0.0066237281124116 | Epoch: 1447 |

MeanAbsoluteError: 0.0562896095216274 | Loss: 0.0065249161592025 | Epoch: 1448 |

MeanAbsoluteError: 0.0565231181681156 | Loss: 0.0057698578926769 | Epoch: 1449 |

MeanAbsoluteError: 0.0595677681267262 | Loss: 0.0070670291615049 | Epoch: 1450 |

MeanAbsoluteError: 0.0566757060587406 | Loss: 0.0064075071495366 | Epoch: 1451 |

MeanAbsoluteError: 0.0553364939987659 | Loss: 0.0060599146505653 | Epoch: 1452 |

MeanAbsoluteError: 0.0541689358651638 | Loss: 0.0056800012578363 | Epoch: 1453 |

MeanAbsoluteError: 0.0576780438423157 | Loss: 0.0062125273360531 | Epoch: 1454 |

MeanAbsoluteError: 0.0583593435585499 | Loss: 0.0059080858032636 | Epoch: 1455 |

MeanAbsoluteError: 0.0580403059720993 | Loss: 0.0062801733252369 | Epoch: 1456 |

MeanAbsoluteError: 0.0542672164738178 | Loss: 0.0054260122053751 | Epoch: 1457 |

MeanAbsoluteError: 0.0566761456429958 | Loss: 0.0053721861707163 | Epoch: 1458 |

MeanAbsoluteError: 0.0600563809275627 | Loss: 0.0072668787061169 | Epoch: 1459 |

MeanAbsoluteError: 0.0601928122341633 | Loss: 0.0066354076128603 | Epoch: 1460 |

MeanAbsoluteError: 0.0615716874599457 | Loss: 0.0072449304795388 | Epoch: 1461 |

MeanAbsoluteError: 0.0623099952936172 | Loss: 0.0071688099633502 | Epoch: 1462 |

MeanAbsoluteError: 0.0587708987295628 | Loss: 0.0062015076691750 | Epoch: 1463 |

MeanAbsoluteError: 0.0552529580891132 | Loss: 0.0051825371613813 | Epoch: 1464 |

MeanAbsoluteError: 0.0548544265329838 | Loss: 0.0059621526295329 | Epoch: 1465 |

MeanAbsoluteError: 0.0582481063902378 | Loss: 0.0062729384214375 | Epoch: 1466 |

MeanAbsoluteError: 0.0540102384984493 | Loss: 0.0055912430056681 | Epoch: 1467 |

```

MeanAbsoluteError: 0.0622882954776287 | Loss: 0.0075493527316454 | Epoch: 1468 |

MeanAbsoluteError: 0.0575348027050495 | Loss: 0.0066178210571858 | Epoch: 1469 |

MeanAbsoluteError: 0.0573052652180195 | Loss: 0.0067825304956811 | Epoch: 1470 |

MeanAbsoluteError: 0.0569934137165546 | Loss: 0.0060027882429373 | Epoch: 1471 |

MeanAbsoluteError: 0.0584323443472385 | Loss: 0.0064521284073029 | Epoch: 1472 |

MeanAbsoluteError: 0.0580769143998623 | Loss: 0.0066684547921795 | Epoch: 1473 |

MeanAbsoluteError: 0.0551889874041080 | Loss: 0.0058412185238558 | Epoch: 1474 |

MeanAbsoluteError: 0.0520878881216049 | Loss: 0.0056936428788564 | Epoch: 1475 |

MeanAbsoluteError: 0.0621255300939083 | Loss: 0.0079296191722460 | Epoch: 1476 |

MeanAbsoluteError: 0.0540063343942165 | Loss: 0.0054856804448415 | Epoch: 1477 |

MeanAbsoluteError: 0.0582624301314354 | Loss: 0.0072453686078128 | Epoch: 1478 |

MeanAbsoluteError: 0.0582592338323593 | Loss: 0.0060820765321720 | Epoch: 1479 |

MeanAbsoluteError: 0.0589326173067093 | Loss: 0.0064547484678769 | Epoch: 1480 |

MeanAbsoluteError: 0.0584228076040745 | Loss: 0.0064737026568825 | Epoch: 1481 |

MeanAbsoluteError: 0.0592902749776840 | Loss: 0.0067171421772218 | Early stopping at epoch 14
Returned to Spot: Validation loss: 0.006717142177221831

config: {'_L_in': 10, '_L_out': 1, 'l1': 16, 'dropout_prob': 0.1773189149831582, 'lr_mult': 9}
Epoch: 1 | MeanAbsoluteError: 0.1381864845752716 | Loss: 0.0297331442420060 | Epoch: 2 |

MeanAbsoluteError: 0.1159845590591431 | Loss: 0.0210684445997079 | Epoch: 3 | MeanAbsoluteError:

```

MeanAbsoluteError: 0.1114576458930969 | Loss: 0.0190360272970671 | Returned to Spot: Validation
config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.3840970624671163, 'lr_mult': 4
Epoch: 1 | MeanAbsoluteError: 0.1367287337779999 | Loss: 0.0290687176104831 | Epoch: 2 | Mean
MeanAbsoluteError: 0.1244917139410973 | Loss: 0.0237616484641637 | Epoch: 4 | MeanAbsoluteError
MeanAbsoluteError: 0.1012638658285141 | Loss: 0.0163994073794272 | Epoch: 7 | MeanAbsoluteError
MeanAbsoluteError: 0.0846344903111458 | Loss: 0.0115213508190783 | Epoch: 11 | MeanAbsoluteError
Epoch: 14 | MeanAbsoluteError: 0.0649687424302101 | Loss: 0.0075180405891795 | Epoch: 15 | Mean
MeanAbsoluteError: 0.0639944300055504 | Loss: 0.0072961616051065 | Epoch: 18 | MeanAbsoluteError
Epoch: 21 | MeanAbsoluteError: 0.0672253146767616 | Loss: 0.0078984906829550 | Epoch: 22 | Mean
MeanAbsoluteError: 0.0676896497607231 | Loss: 0.0076013721868788 | Epoch: 24 | MeanAbsoluteError
Epoch: 27 | MeanAbsoluteError: 0.0643735900521278 | Loss: 0.0066883668462795 | Epoch: 28 | Mean
MeanAbsoluteError: 0.0644125342369080 | Loss: 0.0068537936435620 | Epoch: 30 | MeanAbsoluteError
MeanAbsoluteError: 0.0554178804159164 | Loss: 0.0058718559326065 | Epoch: 34 | MeanAbsoluteError
MeanAbsoluteError: 0.0601697154343128 | Loss: 0.0064660886602819 | Epoch: 37 | MeanAbsoluteError
MeanAbsoluteError: 0.0563287734985352 | Loss: 0.0056473667224820 | Epoch: 40 | MeanAbsoluteError
MeanAbsoluteError: 0.0579698123037815 | Loss: 0.0069268529983482 | Epoch: 43 | MeanAbsoluteError
MeanAbsoluteError: 0.0503163449466228 | Loss: 0.0051259728237423 | Epoch: 47 | MeanAbsoluteError
MeanAbsoluteError: 0.0514329001307487 | Loss: 0.0048348851315064 | Epoch: 50 | MeanAbsoluteError
Epoch: 53 | MeanAbsoluteError: 0.0707381442189217 | Loss: 0.0080786086741443 | Epoch: 54 | Mean

```
MeanAbsoluteError: 0.0560877807438374 | Loss: 0.0053623238498786 | Epoch: 56 | MeanAbsoluteE
MeanAbsoluteError: 0.0533880852162838 | Loss: 0.0051999232573986 | Epoch: 60 | MeanAbsoluteE
MeanAbsoluteError: 0.0630423128604889 | Loss: 0.0066528442119689 | Epoch: 63 | MeanAbsoluteE
Epoch: 66 | MeanAbsoluteError: 0.0605428554117680 | Loss: 0.0059999938361886 | Epoch: 67 | M
MeanAbsoluteError: 0.0520688518881798 | Loss: 0.0049734828475936 | Epoch: 69 | MeanAbsoluteE
MeanAbsoluteError: 0.0587881840765476 | Loss: 0.0057495791238341 | Epoch: 73 | MeanAbsoluteE
MeanAbsoluteError: 0.0478166267275810 | Loss: 0.0041767671385682 | Epoch: 76 | MeanAbsoluteE
MeanAbsoluteError: 0.0519948042929173 | Loss: 0.0049195504352132 | Epoch: 79 | MeanAbsoluteE
MeanAbsoluteError: 0.0500195659697056 | Loss: 0.0051155700378953 | Epoch: 82 | MeanAbsoluteE
MeanAbsoluteError: 0.0515358150005341 | Loss: 0.0041499671106545 | Epoch: 86 | MeanAbsoluteE
MeanAbsoluteError: 0.0500534586608410 | Loss: 0.0048268302971816 | Epoch: 89 | MeanAbsoluteE
Epoch: 92 | MeanAbsoluteError: 0.0520430393517017 | Loss: 0.0047456365764925 | Epoch: 93 | M
MeanAbsoluteError: 0.0530404150485992 | Loss: 0.0045820627404426 | Epoch: 95 | MeanAbsoluteE
MeanAbsoluteError: 0.0548297166824341 | Loss: 0.0050166448008416 | Epoch: 98 | MeanAbsoluteE
MeanAbsoluteError: 0.0548061318695545 | Loss: 0.0052323131132748 | Epoch: 101 | MeanAbsolutel
MeanAbsoluteError: 0.0509681552648544 | Loss: 0.0048525189340580 | Epoch: 104 | MeanAbsolutel
MeanAbsoluteError: 0.0436878465116024 | Loss: 0.0039610523281705 | Epoch: 107 | MeanAbsolutel
MeanAbsoluteError: 0.0496334806084633 | Loss: 0.0041041335207410 | Epoch: 110 | MeanAbsolutel
MeanAbsoluteError: 0.0542057938873768 | Loss: 0.0050549143772131 | Epoch: 113 | MeanAbsolutel
```



```

config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.20293466636782523, 'lr_mult':
Epoch: 1 | MeanAbsoluteError: 0.1481190621852875 | Loss: 0.0351861426116605 | Epoch: 2 | Mean
MeanAbsoluteError: 0.1284587234258652 | Loss: 0.0260072121592729 | Epoch: 7 | MeanAbsoluteError:
MeanAbsoluteError: 0.1256589442491531 | Loss: 0.0231270027395926 | Epoch: 13 | MeanAbsoluteError:
MeanAbsoluteError: 0.1269076168537140 | Loss: 0.0210259428836013 | Epoch: 19 | MeanAbsoluteError:
MeanAbsoluteError: 0.0746236294507980 | Loss: 0.0090405947626813 | Epoch: 25 | MeanAbsoluteError:
MeanAbsoluteError: 0.1480638384819031 | Loss: 0.0272101940293061 | Epoch: 31 | MeanAbsoluteError:
MeanAbsoluteError: 0.0467006415128708 | Loss: 0.0039529639761895 | Epoch: 37 | MeanAbsoluteError:
MeanAbsoluteError: 0.0696488469839096 | Loss: 0.0074459726456553 | Epoch: 43 | MeanAbsoluteError:
MeanAbsoluteError: 0.1641799360513687 | Loss: 0.0306064282592974 | Epoch: 49 | MeanAbsoluteError:
MeanAbsoluteError: 0.0419364310801029 | Loss: 0.0025509907557678 | Epoch: 55 | MeanAbsoluteError:
MeanAbsoluteError: 0.0432921573519707 | Loss: 0.0031951448978170 | Epoch: 61 | MeanAbsoluteError:
MeanAbsoluteError: 0.0494345240294933 | Loss: 0.0046315880761923 | Epoch: 67 | MeanAbsoluteError:
MeanAbsoluteError: 0.0768425613641739 | Loss: 0.0075823354877924 | Epoch: 73 | MeanAbsoluteError:
MeanAbsoluteError: 0.0565533749759197 | Loss: 0.0057808661000117 | Epoch: 79 | MeanAbsoluteError:
MeanAbsoluteError: 0.0412385798990726 | Loss: 0.0032702302437668 | Epoch: 85 | MeanAbsoluteError:
MeanAbsoluteError: 0.0559617988765240 | Loss: 0.0060383668169379 | Epoch: 91 | MeanAbsoluteError:
MeanAbsoluteError: 0.0324185937643051 | Loss: 0.0018319249379841 | Epoch: 97 | MeanAbsoluteError:
MeanAbsoluteError: 0.0661612898111343 | Loss: 0.0073537388081221 | Epoch: 103 | MeanAbsoluteError:

```



```

MeanAbsoluteError: 0.0463805235922337 | Loss: 0.0037776903482154 | Epoch: 49 | MeanAbsoluteError: 0.0463805235922337
MeanAbsoluteError: 0.1056240573525429 | Loss: 0.0132143862153354 | Epoch: 55 | MeanAbsoluteError: 0.1056240573525429
MeanAbsoluteError: 0.0419648997485638 | Loss: 0.0033961875290659 | Epoch: 61 | MeanAbsoluteError: 0.0419648997485638
MeanAbsoluteError: 0.0358048081398010 | Loss: 0.0020942501723766 | Epoch: 67 | MeanAbsoluteError: 0.0358048081398010
MeanAbsoluteError: 0.0528490282595158 | Loss: 0.0043591684416721 | Epoch: 73 | MeanAbsoluteError: 0.0528490282595158
MeanAbsoluteError: 0.0367655195295811 | Loss: 0.0024710077865932 | Epoch: 79 | MeanAbsoluteError: 0.0367655195295811
MeanAbsoluteError: 0.0442682057619095 | Loss: 0.0036167056716390 | Epoch: 85 | MeanAbsoluteError: 0.0442682057619095
MeanAbsoluteError: 0.0513483658432961 | Loss: 0.0044952854269037 | Epoch: 91 | MeanAbsoluteError: 0.0513483658432961
MeanAbsoluteError: 0.0428741872310638 | Loss: 0.0034495269279241 | Epoch: 97 | MeanAbsoluteError: 0.0428741872310638
MeanAbsoluteError: 0.0419684201478958 | Loss: 0.0032739848339636 | Epoch: 103 | MeanAbsoluteError: 0.0419684201478958
MeanAbsoluteError: 0.0389656983315945 | Loss: 0.0024939042755010 | Epoch: 109 | MeanAbsoluteError: 0.0389656983315945
MeanAbsoluteError: 0.0424479246139526 | Loss: 0.0034096448094045 | Epoch: 115 | MeanAbsoluteError: 0.0424479246139526
MeanAbsoluteError: 0.0544564165174961 | Loss: 0.0046442420540476 | Epoch: 121 | MeanAbsoluteError: 0.0544564165174961
Returned to Spot: Validation loss: 0.0020169751859564137

spotPython tuning: 0.0016865455361671354 [####-----] 37.63%

config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.1655924245415988, 'lr_mult': 4}
Epoch: 1 | MeanAbsoluteError: 0.1433156728744507 | Loss: 0.0317698864168242 | Epoch: 2 | MeanAbsoluteError: 0.1433156728744507
MeanAbsoluteError: 0.0936248227953911 | Loss: 0.0137669800897129 | Epoch: 5 | MeanAbsoluteError: 0.0936248227953911
MeanAbsoluteError: 0.1494184434413910 | Loss: 0.0303587605441479 | Epoch: 9 | MeanAbsoluteError: 0.1494184434413910

```

MeanAbsoluteError: 0.1464235931634903		Loss: 0.0270001979201640		Epoch: 13		MeanAbsoluteError: 0.0796796903014183
MeanAbsoluteError: 0.0796796903014183		Loss: 0.0092711056227257		Epoch: 17		MeanAbsoluteError: 0.0889066904783249
MeanAbsoluteError: 0.0889066904783249		Loss: 0.0101355453462977		Epoch: 21		MeanAbsoluteError: 0.0638134703040123
MeanAbsoluteError: 0.0638134703040123		Loss: 0.0060557747247482		Epoch: 25		MeanAbsoluteError: 0.0509321652352810
MeanAbsoluteError: 0.0509321652352810		Loss: 0.0039346609708829		Epoch: 29		MeanAbsoluteError: 0.0892713665962219
MeanAbsoluteError: 0.0892713665962219		Loss: 0.0098597169179763		Epoch: 33		MeanAbsoluteError: 0.0469895265996456
MeanAbsoluteError: 0.0469895265996456		Loss: 0.0043270995504970		Epoch: 37		MeanAbsoluteError: 0.0379331782460213
MeanAbsoluteError: 0.0379331782460213		Loss: 0.0027486923193608		Epoch: 41		MeanAbsoluteError: 0.0473294816911221
MeanAbsoluteError: 0.0473294816911221		Loss: 0.0036540160163943		Epoch: 44		MeanAbsoluteError: 0.0432029701769352
MeanAbsoluteError: 0.0432029701769352		Loss: 0.0034410358042040		Epoch: 47		MeanAbsoluteError: 0.0747778043150902
MeanAbsoluteError: 0.0747778043150902		Loss: 0.0074303244841040		Epoch: 51		MeanAbsoluteError: 0.0555500164628029
MeanAbsoluteError: 0.0555500164628029		Loss: 0.0043365547776614		Epoch: 55		MeanAbsoluteError: 0.0555892698466778
MeanAbsoluteError: 0.0555892698466778		Loss: 0.0045445883297361		Epoch: 59		MeanAbsoluteError: 0.1158791109919548
MeanAbsoluteError: 0.1158791109919548		Loss: 0.0152969174822302		Epoch: 63		MeanAbsoluteError: 0.0401127897202969
MeanAbsoluteError: 0.0401127897202969		Loss: 0.0033251352096589		Epoch: 67		MeanAbsoluteError: 0.0393033884465694
MeanAbsoluteError: 0.0393033884465694		Loss: 0.0031038368898934		Epoch: 71		MeanAbsoluteError: 0.0442191176116467
MeanAbsoluteError: 0.0442191176116467		Loss: 0.0039995709161495		Epoch: 75		MeanAbsoluteError: 0.0298453234136105
MeanAbsoluteError: 0.0298453234136105		Loss: 0.0018974501788187		Epoch: 79		MeanAbsoluteError: 0.0415444262325764
MeanAbsoluteError: 0.0415444262325764		Loss: 0.0030644259547300		Epoch: 83		

MeanAbsoluteError: 0.1320481449365616 | Loss: 0.0284630910150315 | Epoch: 9 | MeanAbsoluteError: 0.1331644356250763 | Loss: 0.0284663193344482 | Epoch: 11 | MeanAbsoluteError: 0.1328123658895493 | Loss: 0.0286446425107945 | Epoch: 13 | MeanAbsoluteError: 0.1266266107559204 | Loss: 0.0262581179482176 | Epoch: 15 | MeanAbsoluteError: 0.1278899908065796 | Loss: 0.0271047620151780 | Epoch: 17 | MeanAbsoluteError: 0.1264368742704391 | Loss: 0.0259645239719631 | Epoch: 19 | MeanAbsoluteError: 0.1276822090148926 | Loss: 0.0258632998018967 | Epoch: 21 | MeanAbsoluteError: 0.1253378093242645 | Loss: 0.0249380984123012 | Epoch: 23 | MeanAbsoluteError: 0.1243551746010780 | Loss: 0.0252211378323600 | Epoch: 25 | MeanAbsoluteError: 0.1205481588840485 | Loss: 0.0245629931726542 | Epoch: 27 | MeanAbsoluteError: 0.1214140951633453 | Loss: 0.0248758095243063 | Epoch: 29 | MeanAbsoluteError: 0.1156355887651443 | Loss: 0.0233179663038371 | Epoch: 31 | MeanAbsoluteError: 0.1186462789773941 | Loss: 0.0231075846884204 | Epoch: 33 | MeanAbsoluteError: 0.1162924692034721 | Loss: 0.0217907584983071 | Epoch: 35 | MeanAbsoluteError: 0.1136435940861702 | Loss: 0.0200268344304181 | Epoch: 37 | MeanAbsoluteError: 0.1070275977253914 | Loss: 0.0191006068336336 | Epoch: 39 | MeanAbsoluteError: 0.1075686365365982 | Loss: 0.0198765659925381 | Epoch: 41 | MeanAbsoluteError: 0.1103932932019234 | Loss: 0.0200265646654818 | Epoch: 43 | MeanAbsoluteError: 0.1068583279848099 | Loss: 0.0180320290076595 | Epoch: 45 | MeanAbsoluteError:

MeanAbsoluteError: 0.1014720201492310 | Loss: 0.0174357497966603 | Epoch: 47 | MeanAbsoluteError: 0.1052749082446098 | Loss: 0.0182511263196111 | Epoch: 49 | MeanAbsoluteError: 0.1040564924478531 | Loss: 0.0176442014849990 | Epoch: 51 | MeanAbsoluteError: 0.0925017371773720 | Loss: 0.0144119317226700 | Epoch: 53 | MeanAbsoluteError: 0.0995326265692711 | Loss: 0.0157204835038436 | Epoch: 55 | MeanAbsoluteError: 0.0947979390621185 | Loss: 0.0148047510546779 | Epoch: 57 | MeanAbsoluteError: 0.0888242945075035 | Loss: 0.0128873252481418 | Epoch: 59 | MeanAbsoluteError: 0.0951983034610748 | Loss: 0.0152086828051037 | Epoch: 61 | MeanAbsoluteError: 0.0886269956827164 | Loss: 0.0128626695092718 | Epoch: 63 | MeanAbsoluteError: 0.0891181305050850 | Loss: 0.0128270530472755 | Epoch: 65 | MeanAbsoluteError: 0.0902533009648323 | Loss: 0.0131731714538642 | Epoch: 67 | MeanAbsoluteError: 0.0853529199957848 | Loss: 0.0123007992362162 | Epoch: 69 | MeanAbsoluteError: 0.0876831561326981 | Loss: 0.0131920555385908 | Epoch: 71 | MeanAbsoluteError: 0.0888556614518166 | Loss: 0.0121744081108390 | Epoch: 73 | MeanAbsoluteError: 0.0820376724004745 | Loss: 0.0112578649668170 | Epoch: 75 | MeanAbsoluteError: 0.0820805728435516 | Loss: 0.0116065514188162 | Epoch: 77 | MeanAbsoluteError: 0.0813824236392975 | Loss: 0.0119170993382699 | Epoch: 79 | MeanAbsoluteError: 0.0822135210037231 | Loss: 0.0117135195116709 | Epoch: 81 | MeanAbsoluteError: 0.0936757251620293 | Loss: 0.0143759323308538 | Epoch: 83 | MeanAbsoluteError:

MeanAbsoluteError:	0.0764572620391846		Loss:	0.0096807446680032		Epoch:	85		MeanAbsoluteError:	0.0764572620391846
MeanAbsoluteError:	0.0718123167753220		Loss:	0.0090759093352397		Epoch:	87		MeanAbsoluteError:	0.0718123167753220
MeanAbsoluteError:	0.0735909268260002		Loss:	0.0105060545641831		Epoch:	89		MeanAbsoluteError:	0.0735909268260002
MeanAbsoluteError:	0.0753822103142738		Loss:	0.0096769084553479		Epoch:	91		MeanAbsoluteError:	0.0753822103142738
MeanAbsoluteError:	0.0734286531805992		Loss:	0.0094129407294340		Epoch:	93		MeanAbsoluteError:	0.0734286531805992
MeanAbsoluteError:	0.0782692506909370		Loss:	0.0112351490411368		Epoch:	95		MeanAbsoluteError:	0.0782692506909370
MeanAbsoluteError:	0.0776935964822769		Loss:	0.0101845112175828		Epoch:	97		MeanAbsoluteError:	0.0776935964822769
MeanAbsoluteError:	0.0721559301018715		Loss:	0.0088397628646099		Epoch:	99		MeanAbsoluteError:	0.0721559301018715
MeanAbsoluteError:	0.0749300047755241		Loss:	0.0094689225876018		Epoch:	101		MeanAbsoluteError:	0.0749300047755241
MeanAbsoluteError:	0.0701709911227226		Loss:	0.0088093907474295		Epoch:	103		MeanAbsoluteError:	0.0701709911227226
MeanAbsoluteError:	0.0707982033491135		Loss:	0.0088555532336039		Epoch:	105		MeanAbsoluteError:	0.0707982033491135
MeanAbsoluteError:	0.0733223631978035		Loss:	0.0096679325766959		Epoch:	107		MeanAbsoluteError:	0.0733223631978035
MeanAbsoluteError:	0.0678948163986206		Loss:	0.0080376295066488		Epoch:	109		MeanAbsoluteError:	0.0678948163986206
MeanAbsoluteError:	0.0639514625072479		Loss:	0.0071893201185096		Epoch:	111		MeanAbsoluteError:	0.0639514625072479
MeanAbsoluteError:	0.0664180219173431		Loss:	0.0076895217552144		Epoch:	113		MeanAbsoluteError:	0.0664180219173431
MeanAbsoluteError:	0.0694915726780891		Loss:	0.0073349445470070		Epoch:	115		MeanAbsoluteError:	0.0694915726780891
MeanAbsoluteError:	0.0612682551145554		Loss:	0.0065525298383651		Epoch:	117		MeanAbsoluteError:	0.0612682551145554
MeanAbsoluteError:	0.0758399739861488		Loss:	0.0096607358121362		Epoch:	119		MeanAbsoluteError:	0.0758399739861488
MeanAbsoluteError:	0.0692536160349846		Loss:	0.0087750128433598		Epoch:	121		MeanAbsoluteError:	0.0692536160349846

MeanAbsoluteError:	0.0593885853886604		Loss:	0.0064107698792788		Epoch:	123		MeanAbsoluteError:
MeanAbsoluteError:	0.0632705017924309		Loss:	0.0070930818573719		Epoch:	125		MeanAbsoluteError:
MeanAbsoluteError:	0.0644011870026588		Loss:	0.0081850338610820		Epoch:	127		MeanAbsoluteError:
MeanAbsoluteError:	0.0675194710493088		Loss:	0.0078604066082718		Epoch:	129		MeanAbsoluteError:
MeanAbsoluteError:	0.0604964047670364		Loss:	0.0065711106320745		Epoch:	131		MeanAbsoluteError:
MeanAbsoluteError:	0.0649345144629478		Loss:	0.0067558406980855		Epoch:	133		MeanAbsoluteError:
MeanAbsoluteError:	0.0658221319317818		Loss:	0.0082103622017281		Epoch:	135		MeanAbsoluteError:
MeanAbsoluteError:	0.0692525729537010		Loss:	0.0083139725305189		Epoch:	137		MeanAbsoluteError:
MeanAbsoluteError:	0.0639452263712883		Loss:	0.0079015905227463		Epoch:	139		MeanAbsoluteError:
MeanAbsoluteError:	0.0683363378047943		Loss:	0.0084703465016853		Epoch:	141		MeanAbsoluteError:
MeanAbsoluteError:	0.0615076385438442		Loss:	0.0065451042763399		Epoch:	143		MeanAbsoluteError:
MeanAbsoluteError:	0.0652786418795586		Loss:	0.0077757252821405		Epoch:	145		MeanAbsoluteError:
MeanAbsoluteError:	0.0575554482638836		Loss:	0.0060203835616305		Epoch:	147		MeanAbsoluteError:
MeanAbsoluteError:	0.0622477009892464		Loss:	0.0069021634806536		Epoch:	149		MeanAbsoluteError:
MeanAbsoluteError:	0.0619794391095638		Loss:	0.0073528092578176		Epoch:	151		MeanAbsoluteError:
MeanAbsoluteError:	0.0597691610455513		Loss:	0.0068328659422389		Epoch:	153		MeanAbsoluteError:
MeanAbsoluteError:	0.0578564777970314		Loss:	0.0069699554189451		Epoch:	155		MeanAbsoluteError:
MeanAbsoluteError:	0.0720038339495659		Loss:	0.0086964950100274		Epoch:	157		MeanAbsoluteError:
MeanAbsoluteError:	0.0610565952956676		Loss:	0.0064982056133694		Epoch:	159		MeanAbsoluteError:

MeanAbsoluteError:	0.0696950033307076		Loss:	0.0083177802704373		Epoch:	161		MeanAbsoluteError:
MeanAbsoluteError:	0.0628887116909027		Loss:	0.0075323162569762		Epoch:	163		MeanAbsoluteError:
MeanAbsoluteError:	0.0617038905620575		Loss:	0.0069494730805194		Epoch:	165		MeanAbsoluteError:
MeanAbsoluteError:	0.0585346594452858		Loss:	0.0058759608502059		Epoch:	167		MeanAbsoluteError:
MeanAbsoluteError:	0.0634653046727180		Loss:	0.0072934784448559		Epoch:	169		MeanAbsoluteError:
MeanAbsoluteError:	0.0579071678221226		Loss:	0.0066468201372405		Epoch:	171		MeanAbsoluteError:
MeanAbsoluteError:	0.0743187740445137		Loss:	0.0091092717178215		Epoch:	173		MeanAbsoluteError:
MeanAbsoluteError:	0.0573926866054535		Loss:	0.0059074133406332		Epoch:	175		MeanAbsoluteError:
MeanAbsoluteError:	0.0579331219196320		Loss:	0.0069481165096237		Epoch:	177		MeanAbsoluteError:
MeanAbsoluteError:	0.0591617822647095		Loss:	0.0056288513036347		Epoch:	179		MeanAbsoluteError:
MeanAbsoluteError:	0.0724572986364365		Loss:	0.0088172616424823		Epoch:	181		MeanAbsoluteError:
MeanAbsoluteError:	0.0518214553594589		Loss:	0.0047711538285703		Epoch:	183		MeanAbsoluteError:
MeanAbsoluteError:	0.0511371530592442		Loss:	0.0049053175822145		Epoch:	185		MeanAbsoluteError:
MeanAbsoluteError:	0.0560934767127037		Loss:	0.0056336540790079		Epoch:	187		MeanAbsoluteError:
MeanAbsoluteError:	0.0650602206587791		Loss:	0.0074197149369866		Epoch:	189		MeanAbsoluteError:
MeanAbsoluteError:	0.0494736246764660		Loss:	0.0042421762543534		Epoch:	191		MeanAbsoluteError:
MeanAbsoluteError:	0.0550707168877125		Loss:	0.0059105275891182		Epoch:	193		MeanAbsoluteError:
MeanAbsoluteError:	0.0583632215857506		Loss:	0.0055343655066712		Epoch:	195		MeanAbsoluteError:
MeanAbsoluteError:	0.0519710183143616		Loss:	0.0052383702668043		Epoch:	197		MeanAbsoluteError:

MeanAbsoluteError:	0.0547879673540592		Loss:	0.0060135620633924		Epoch:	199		MeanAbsoluteError:
MeanAbsoluteError:	0.0632351413369179		Loss:	0.0070556181507479		Epoch:	201		MeanAbsoluteError:
MeanAbsoluteError:	0.0524623803794384		Loss:	0.0047075584098868		Epoch:	203		MeanAbsoluteError:
MeanAbsoluteError:	0.0592683143913746		Loss:	0.0067987406846920		Epoch:	205		MeanAbsoluteError:
MeanAbsoluteError:	0.0576860681176186		Loss:	0.0067315095925312		Epoch:	207		MeanAbsoluteError:
MeanAbsoluteError:	0.0489558130502701		Loss:	0.0052356466323162		Epoch:	209		MeanAbsoluteError:
MeanAbsoluteError:	0.0547465123236179		Loss:	0.0058934181885745		Epoch:	211		MeanAbsoluteError:
MeanAbsoluteError:	0.0552970394492149		Loss:	0.0060856258466007		Epoch:	213		MeanAbsoluteError:
MeanAbsoluteError:	0.0551117844879627		Loss:	0.0051995676813500		Epoch:	215		MeanAbsoluteError:
MeanAbsoluteError:	0.0556403659284115		Loss:	0.0055973850447979		Epoch:	217		MeanAbsoluteError:
MeanAbsoluteError:	0.0572885498404503		Loss:	0.0066044491587060		Epoch:	219		MeanAbsoluteError:
MeanAbsoluteError:	0.0500331409275532		Loss:	0.0050221494978024		Epoch:	221		MeanAbsoluteError:
MeanAbsoluteError:	0.0574217289686203		Loss:	0.0059297246993274		Epoch:	223		MeanAbsoluteError:
MeanAbsoluteError:	0.0549992173910141		Loss:	0.0062409570060768		Epoch:	225		MeanAbsoluteError:
MeanAbsoluteError:	0.0616002082824707		Loss:	0.0067100787755886		Epoch:	227		MeanAbsoluteError:
MeanAbsoluteError:	0.0522722937166691		Loss:	0.0047962576599706		Epoch:	229		MeanAbsoluteError:
MeanAbsoluteError:	0.0679422765970230		Loss:	0.0073958520569201		Epoch:	231		MeanAbsoluteError:
MeanAbsoluteError:	0.0478421598672867		Loss:	0.0043366839343339		Epoch:	233		MeanAbsoluteError:
MeanAbsoluteError:	0.0495321415364742		Loss:	0.0045163590586622		Epoch:	235		MeanAbsoluteError:

MeanAbsoluteError:	0.0506273731589317		Loss:	0.0048551004989636		Epoch:	237		MeanAbsoluteError:
MeanAbsoluteError:	0.0518819950520992		Loss:	0.0051837380068671		Epoch:	239		MeanAbsoluteError:
MeanAbsoluteError:	0.0495546758174896		Loss:	0.0050656861643993		Epoch:	241		MeanAbsoluteError:
MeanAbsoluteError:	0.0561373531818390		Loss:	0.0059761182791063		Epoch:	243		MeanAbsoluteError:
MeanAbsoluteError:	0.0624603889882565		Loss:	0.0064177193470593		Epoch:	245		MeanAbsoluteError:
MeanAbsoluteError:	0.0540331527590752		Loss:	0.0052537596784532		Epoch:	247		MeanAbsoluteError:
MeanAbsoluteError:	0.0509700886905193		Loss:	0.0057799126156361		Epoch:	249		MeanAbsoluteError:
MeanAbsoluteError:	0.0473000556230545		Loss:	0.0047825003857724		Epoch:	251		MeanAbsoluteError:
MeanAbsoluteError:	0.0557649023830891		Loss:	0.0055984221919636		Epoch:	253		MeanAbsoluteError:
MeanAbsoluteError:	0.0472091883420944		Loss:	0.0044437498620123		Epoch:	255		MeanAbsoluteError:
MeanAbsoluteError:	0.0511254072189331		Loss:	0.0045950685499089		Epoch:	257		MeanAbsoluteError:
MeanAbsoluteError:	0.0530676916241646		Loss:	0.0054226482683780		Epoch:	259		MeanAbsoluteError:
MeanAbsoluteError:	0.0559468008577824		Loss:	0.0052178795725156		Epoch:	261		MeanAbsoluteError:
MeanAbsoluteError:	0.0540579855442047		Loss:	0.0055207787449227		Epoch:	263		MeanAbsoluteError:
MeanAbsoluteError:	0.0546494945883751		Loss:	0.0054079308205186		Epoch:	265		MeanAbsoluteError:
MeanAbsoluteError:	0.0539024993777275		Loss:	0.0051196760553131		Epoch:	267		MeanAbsoluteError:
MeanAbsoluteError:	0.0540158599615097		Loss:	0.0056552773029053		Epoch:	269		MeanAbsoluteError:
MeanAbsoluteError:	0.0471969135105610		Loss:	0.0043652107982014		Epoch:	271		MeanAbsoluteError:
MeanAbsoluteError:	0.0458981022238731		Loss:	0.0044028544430866		Epoch:	273		MeanAbsoluteError:

MeanAbsoluteError:	0.0485126972198486		Loss:	0.0047815723090408		Epoch:	275		MeanAbsoluteError:
MeanAbsoluteError:	0.0464547649025917		Loss:	0.0045896124333682		Epoch:	277		MeanAbsoluteError:
MeanAbsoluteError:	0.0486301183700562		Loss:	0.0047406154412065		Epoch:	279		MeanAbsoluteError:
MeanAbsoluteError:	0.0498691461980343		Loss:	0.0047164251430475		Epoch:	281		MeanAbsoluteError:
MeanAbsoluteError:	0.0503310821950436		Loss:	0.0052635660042717		Epoch:	283		MeanAbsoluteError:
MeanAbsoluteError:	0.0492931529879570		Loss:	0.0048070204410530		Epoch:	285		MeanAbsoluteError:
MeanAbsoluteError:	0.0493427217006683		Loss:	0.0040915920215316		Epoch:	287		MeanAbsoluteError:
MeanAbsoluteError:	0.0492741465568542		Loss:	0.0048880002654378		Epoch:	289		MeanAbsoluteError:
MeanAbsoluteError:	0.0515524111688137		Loss:	0.0050615872023627		Epoch:	291		MeanAbsoluteError:
MeanAbsoluteError:	0.0476048551499844		Loss:	0.0041292278941623		Epoch:	293		MeanAbsoluteError:
MeanAbsoluteError:	0.0501916520297527		Loss:	0.0048103639473036		Epoch:	295		MeanAbsoluteError:
MeanAbsoluteError:	0.0458452068269253		Loss:	0.0040243154201706		Epoch:	297		MeanAbsoluteError:
MeanAbsoluteError:	0.0435664989054203		Loss:	0.0031693628636851		Epoch:	299		MeanAbsoluteError:
MeanAbsoluteError:	0.0472872965037823		Loss:	0.0039366024782219		Epoch:	301		MeanAbsoluteError:
MeanAbsoluteError:	0.0497904904186726		Loss:	0.0045722577947584		Epoch:	303		MeanAbsoluteError:
MeanAbsoluteError:	0.0470734611153603		Loss:	0.0042264651703207		Epoch:	305		MeanAbsoluteError:
MeanAbsoluteError:	0.0470329262316227		Loss:	0.0050604320904791		Epoch:	307		MeanAbsoluteError:
MeanAbsoluteError:	0.0479018203914165		Loss:	0.0048072240391967		Epoch:	309		MeanAbsoluteError:
MeanAbsoluteError:	0.0447205603122711		Loss:	0.0042268326306887		Epoch:	311		MeanAbsoluteError:

MeanAbsoluteError:	0.0459820255637169		Loss:	0.0037818311048843		Epoch:	313		MeanAbsoluteError:
MeanAbsoluteError:	0.0461256243288517		Loss:	0.0039698160284101		Epoch:	315		MeanAbsoluteError:
MeanAbsoluteError:	0.0538097955286503		Loss:	0.0051410642918199		Epoch:	317		MeanAbsoluteError:
MeanAbsoluteError:	0.0443462319672108		Loss:	0.0040630505567319		Epoch:	319		MeanAbsoluteError:
MeanAbsoluteError:	0.0460803173482418		Loss:	0.0043025450477695		Epoch:	321		MeanAbsoluteError:
MeanAbsoluteError:	0.0511639229953289		Loss:	0.0054869270074720		Epoch:	323		MeanAbsoluteError:
MeanAbsoluteError:	0.0468969196081161		Loss:	0.0047997396583301		Epoch:	325		MeanAbsoluteError:
MeanAbsoluteError:	0.0469854697585106		Loss:	0.0041551016612701		Epoch:	327		MeanAbsoluteError:
MeanAbsoluteError:	0.0457778498530388		Loss:	0.0042384732284853		Epoch:	329		MeanAbsoluteError:
MeanAbsoluteError:	0.0486002862453461		Loss:	0.0045494551764262		Epoch:	331		MeanAbsoluteError:
MeanAbsoluteError:	0.0527529083192348		Loss:	0.0047984617321115		Epoch:	333		MeanAbsoluteError:
MeanAbsoluteError:	0.0541633367538452		Loss:	0.0063094982025704		Epoch:	335		MeanAbsoluteError:
MeanAbsoluteError:	0.0425396077334881		Loss:	0.0037189976673720		Epoch:	337		MeanAbsoluteError:
MeanAbsoluteError:	0.0472558438777924		Loss:	0.0040682157153864		Epoch:	339		MeanAbsoluteError:
MeanAbsoluteError:	0.0456587076187134		Loss:	0.0038851029561269		Epoch:	341		MeanAbsoluteError:
MeanAbsoluteError:	0.0447536259889603		Loss:	0.0036915059042114		Epoch:	343		MeanAbsoluteError:
MeanAbsoluteError:	0.0472966581583023		Loss:	0.0045352896666277		Epoch:	345		MeanAbsoluteError:
MeanAbsoluteError:	0.0594302378594875		Loss:	0.0060798001573666		Epoch:	347		MeanAbsoluteError:
MeanAbsoluteError:	0.0552905388176441		Loss:	0.0057069955203731		Epoch:	349		MeanAbsoluteError:


```

MeanAbsoluteError: 0.1275248080492020 | Loss: 0.0262124702628506 | Epoch: 37 | MeanAbsoluteE
MeanAbsoluteError: 0.1313838213682175 | Loss: 0.0268957358784974 | Epoch: 41 | MeanAbsoluteE
MeanAbsoluteError: 0.1249709948897362 | Loss: 0.0257774698670561 | Epoch: 45 | MeanAbsoluteE
MeanAbsoluteError: 0.1212792992591858 | Loss: 0.0238714892123091 | Epoch: 49 | MeanAbsoluteE
MeanAbsoluteError: 0.1205148175358772 | Loss: 0.0233872304670513 | Epoch: 53 | MeanAbsoluteE
MeanAbsoluteError: 0.1156745627522469 | Loss: 0.0212055349869556 | Epoch: 57 | MeanAbsoluteE
MeanAbsoluteError: 0.1213038787245750 | Loss: 0.0232084330514466 | Epoch: 61 | MeanAbsoluteE
MeanAbsoluteError: 0.1202467978000641 | Loss: 0.0240537151326670 | Returned to Spot: Validat
spotPython tuning: 0.0014845179156461534 [#####] 100.00% Done...

<spotPython.spot.spot.Spot at 0x1842f6da0>

```

19.9 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section [14.9](#), see also the description in the documentation: [Tensorboard](#).

19.10 Step 10: Results

After the hyperparameter tuning run is finished, the results can be analyzed as described in Section [14.10](#).

```

spot_tuner.plot_progress(log_y=False,
    filename="./figures/" + experiment_name+"_progress.png")

```

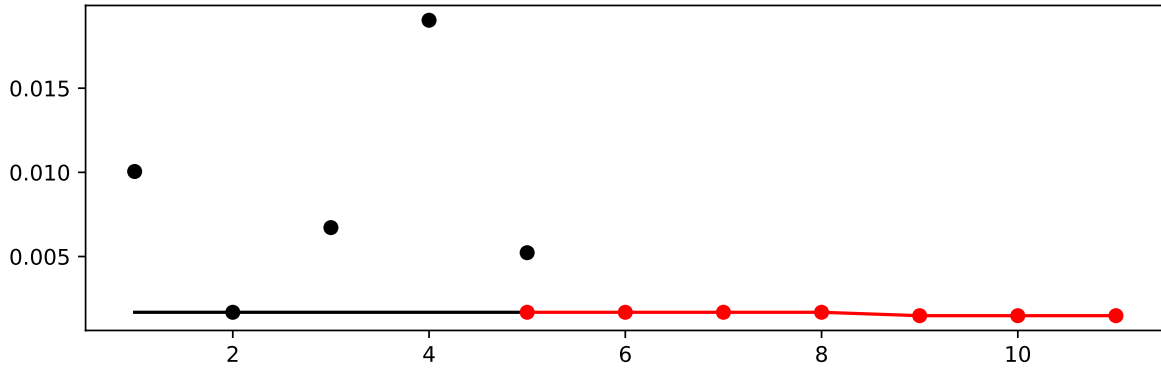


Figure 19.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

```
print(gen_design_table(fun_control=fun_control, spot=spot_tuner))
```

name	type	default	lower	upper	tuned	transform
-----	-----	-----	-----	-----	-----	-----
_L_in	int	10	10.0	10.0	10.0	None
_L_out	int	1	1.0	1.0	1.0	None
l1	int	3	3.0	8.0	5.0	transform_po
dropout_prob	float	0.01	0.0	0.9	0.1655924245415988	None
lr_mult	float	1.0	0.1	10.0	4.186124343189683	None
batch_size	int	4	1.0	4.0	3.0	transform_po
epochs	int	4	2.0	16.0	12.0	transform_po
k_folds	int	1	1.0	1.0	1.0	None
patience	int	2	3.0	7.0	5.0	transform_po
optimizer	factor	SGD	0.0	6.0	0.0	None
sgd_momentum	float	0.0	0.0	1.0	0.07400725452261472	None

```
spot_tuner.plot_importance(threshold=0.025,
                           filename="./figures/" + experiment_name+"_importance.png")
```

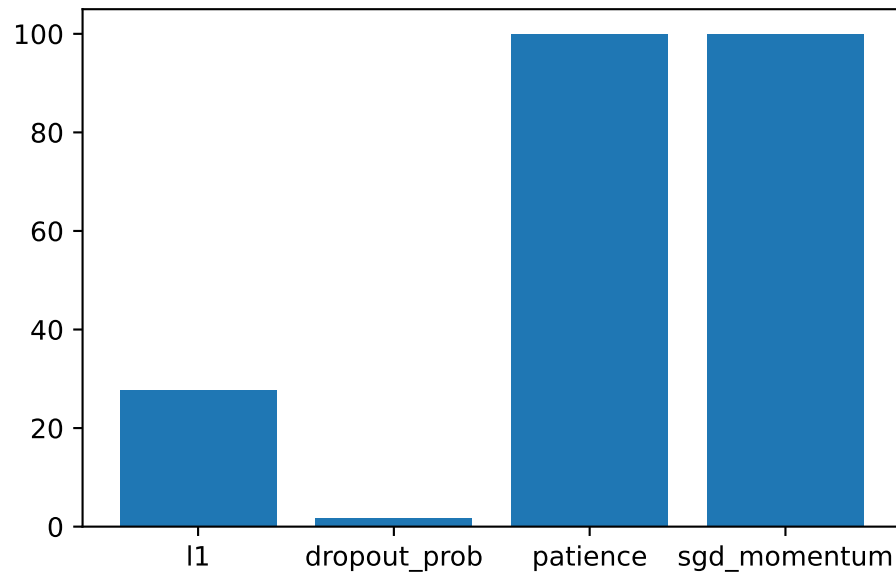


Figure 19.2: Variable importance plot, threshold 0.025.

19.10.1 Get the Tuned Architecture (SPOT Results)

```
from spotPython.hyperparameters.values import get_one_core_model_from_X
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
model_spot = get_one_core_model_from_X(X, fun_control)
model_spot
```

```
Net_lin_reg(
  (fc1): Linear(in_features=10, out_features=32, bias=True)
  (fc2): Linear(in_features=32, out_features=16, bias=True)
  (fc3): Linear(in_features=16, out_features=1, bias=True)
  (relu): ReLU()
  (softmax): Softmax(dim=1)
  (dropout1): Dropout(p=0.1655924245415988, inplace=False)
  (dropout2): Dropout(p=0.0827962122707994, inplace=False)
)
```

19.10.2 Evaluation of the Tuned Architecture

```
from spotPython.torch.traintest import (
    train_tuned,
    test_tuned,
)

train_tuned(net=model_spot, train_dataset=train,
            loss_function=fun_control["loss_function"],
            metric=fun_control["metric_torch"],
            shuffle=True,
            device = fun_control["device"],
            path=None,
            task=fun_control["task"],)
```

```
Epoch: 1 | MeanAbsoluteError: 0.1678686439990997 | Loss: 0.0415245869049900 | Epoch: 2 | Mean
MeanAbsoluteError: 0.1346424818038940 | Loss: 0.0288674441950494 | Epoch: 5 | MeanAbsoluteErr
MeanAbsoluteError: 0.0920179784297943 | Loss: 0.0133763144687308 | Epoch: 9 | MeanAbsoluteErr
MeanAbsoluteError: 0.0740733444690704 | Loss: 0.0089116764812436 | Epoch: 13 | MeanAbsoluteErr
MeanAbsoluteError: 0.1216661483049393 | Loss: 0.0185365356308849 | Epoch: 17 | MeanAbsoluteErr
MeanAbsoluteError: 0.1171517297625542 | Loss: 0.0178285199340041 | Epoch: 21 | MeanAbsoluteErr
MeanAbsoluteError: 0.0616878718137741 | Loss: 0.0063272681467137 | Epoch: 25 | MeanAbsoluteErr
MeanAbsoluteError: 0.0584662519395351 | Loss: 0.0053630514670850 | Epoch: 29 | MeanAbsoluteErr
MeanAbsoluteError: 0.0597739778459072 | Loss: 0.0047285030105788 | Epoch: 33 | MeanAbsoluteErr
MeanAbsoluteError: 0.0381345413625240 | Loss: 0.0021513511080564 | Epoch: 37 | MeanAbsoluteErr
MeanAbsoluteError: 0.0856875330209732 | Loss: 0.0096293318967678 | Epoch: 41 | MeanAbsoluteErr
MeanAbsoluteError: 0.0366703495383263 | Loss: 0.0026067191172747 | Epoch: 45 | MeanAbsoluteErr
```


MeanAbsoluteError: 0.0491634830832481 | Loss: 0.0037221840497008 | Epoch: 125 | MeanAbsoluteError: 0.0329865291714668 | Loss: 0.00181444440170880 | Epoch: 129 | MeanAbsoluteError: 0.0347967259585857 | Loss: 0.0018465771718184 | Early stopping at epoch 130
Returned to Spot: Validation loss: 0.0018465771718183532

If path is set to a filename, e.g., path = "model_spot_trained.pt", the weights of the trained model will be loaded from this file.

```
test_tuned(net=model_spot, test_dataset=test,
            shuffle=False,
            loss_function=fun_control["loss_function"],
            metric=fun_control["metric_torch"],
            device = fun_control["device"],
            task=fun_control["task"],)
```

MeanAbsoluteError: 0.0362048856914043 | Loss: 0.0019534178973117 | Final evaluation: Validation loss: 0.0019534178973117378, nan, tensor(0.0362)
Final evaluation: Validation metric: 0.03620488569140434

19.10.3 Cross-validated Evaluations

- This is the evaluation that will be used in the comparison (evaluatecv has to be updated before, to get metric vlaues!):

```
from spotPython.torch.traintest import evaluate_cv
# modify k-kolds:
setattr(model_spot, "k_folds", 10)
df_eval, df_preds, df_metrics = evaluate_cv(net=model_spot,
                                             dataset=fun_control["data"],
                                             loss_function=fun_control["loss_function"],
                                             metric=fun_control["metric_torch"],
                                             task=fun_control["task"],
                                             writer=fun_control["writer"],
                                             writerId="model_spot_cv",
                                             device = fun_control["device"])
```

Fold: 1
Epoch: 1 | MeanAbsoluteError: 0.1618047505617142 | Loss: 0.0394422174789585 | Epoch: 2 |

MeanAbsoluteError: 0.0948168635368347 | Loss: 0.0141746504590488 | Epoch: 3 | MeanAbsoluteError:

MeanAbsoluteError: 0.0913791954517365 | Loss: 0.0127206999235428 | Epoch: 5 | MeanAbsoluteError:

MeanAbsoluteError: 0.0670675933361053 | Loss: 0.0079651997388842 | Epoch: 7 | MeanAbsoluteError:

Epoch: 9 | MeanAbsoluteError: 0.0542722977697849 | Loss: 0.0051638167441035 | Epoch: 10 |

MeanAbsoluteError: 0.0775565877556801 | Loss: 0.0098437746592726 | Epoch: 11 | MeanAbsoluteError:

MeanAbsoluteError: 0.0553506389260292 | Loss: 0.0042343608802184 | Epoch: 13 | MeanAbsoluteError:

MeanAbsoluteError: 0.0724197700619698 | Loss: 0.0086849843724989 | Epoch: 15 | MeanAbsoluteError:

MeanAbsoluteError: 0.0443972237408161 | Loss: 0.0034246930870442 | Epoch: 17 | MeanAbsoluteError:

MeanAbsoluteError: 0.0570652782917023 | Loss: 0.0047597716060969 | Epoch: 19 | MeanAbsoluteError:

MeanAbsoluteError: 0.0432649999856949 | Loss: 0.0029426976644362 | Epoch: 21 | MeanAbsoluteError:

MeanAbsoluteError: 0.0645979940891266 | Loss: 0.0059385592153726 | Epoch: 23 | MeanAbsoluteError:

MeanAbsoluteError: 0.0365579873323441 | Loss: 0.0024502074005655 | Epoch: 25 | MeanAbsoluteError:

MeanAbsoluteError: 0.0322105512022972 | Loss: 0.0017122150479386 | Epoch: 27 | MeanAbsoluteError:

MeanAbsoluteError: 0.0573455579578876 | Loss: 0.0051780047002607 | Epoch: 29 | MeanAbsoluteError:

MeanAbsoluteError: 0.0529483743011951 | Loss: 0.0053945709861672 | Epoch: 31 | MeanAbsoluteError:

MeanAbsoluteError: 0.0374334193766117 | Loss: 0.0026350526649577 | Epoch: 33 | MeanAbsoluteError:

MeanAbsoluteError: 0.0649197697639465 | Loss: 0.0066424999744273 | Epoch: 35 | MeanAbsoluteError:

MeanAbsoluteError:	0.0379110276699066		Loss:	0.0021584174604728		Epoch:	37		MeanAbsoluteError:	0.0379110276699066
MeanAbsoluteError:	0.0689054951071739		Loss:	0.0069304433328888		Epoch:	39		MeanAbsoluteError:	0.0689054951071739
MeanAbsoluteError:	0.0774688571691513		Loss:	0.0082886646358440		Epoch:	41		MeanAbsoluteError:	0.0774688571691513
MeanAbsoluteError:	0.0803026407957077		Loss:	0.0073229097761214		Epoch:	43		MeanAbsoluteError:	0.0803026407957077
MeanAbsoluteError:	0.0545536242425442		Loss:	0.0049339817353309		Epoch:	45		MeanAbsoluteError:	0.0545536242425442
MeanAbsoluteError:	0.0491816811263561		Loss:	0.0031038886562993		Epoch:	47		MeanAbsoluteError:	0.0491816811263561
MeanAbsoluteError:	0.0338786877691746		Loss:	0.0020380532480955		Epoch:	49		MeanAbsoluteError:	0.0338786877691746
MeanAbsoluteError:	0.0299410913139582		Loss:	0.0017625350290193		Epoch:	51		MeanAbsoluteError:	0.0299410913139582
MeanAbsoluteError:	0.0308872535824776		Loss:	0.0018491343989109		Epoch:	53		MeanAbsoluteError:	0.0308872535824776
MeanAbsoluteError:	0.0261515323072672		Loss:	0.0012303415927678		Epoch:	55		MeanAbsoluteError:	0.0261515323072672
MeanAbsoluteError:	0.0305423717945814		Loss:	0.0020965277318073		Epoch:	57		MeanAbsoluteError:	0.0305423717945814
MeanAbsoluteError:	0.0310138911008835		Loss:	0.0021432984324817		Epoch:	59		MeanAbsoluteError:	0.0310138911008835
MeanAbsoluteError:	0.0278909578919411		Loss:	0.0016212499109455		Epoch:	61		MeanAbsoluteError:	0.0278909578919411
MeanAbsoluteError:	0.0358627326786518		Loss:	0.0019277437464692		Epoch:	63		MeanAbsoluteError:	0.0358627326786518
MeanAbsoluteError:	0.0428690947592258		Loss:	0.0028987330778574		Epoch:	65		MeanAbsoluteError:	0.0428690947592258
MeanAbsoluteError:	0.0324590206146240		Loss:	0.0020066216820851		Epoch:	67		MeanAbsoluteError:	0.0324590206146240
MeanAbsoluteError:	0.0372074879705906		Loss:	0.0019988693887941		Epoch:	69		MeanAbsoluteError:	0.0372074879705906
MeanAbsoluteError:	0.0469849854707718		Loss:	0.0033260236064402		Epoch:	71		MeanAbsoluteError:	0.0469849854707718
MeanAbsoluteError:	0.0613652132451534		Loss:	0.0043922790791839		Epoch:	73		MeanAbsoluteError:	0.0613652132451534

```
MeanAbsoluteError: 0.0326935388147831 | Loss: 0.0021480596534765 | Epoch: 75 | MeanAbsoluteE  
MeanAbsoluteError: 0.0729192495346069 | Loss: 0.0069603923564920 | Epoch: 77 | MeanAbsoluteE  
MeanAbsoluteError: 0.0352767892181873 | Loss: 0.0017835934682248 | Epoch: 79 | MeanAbsoluteE  
MeanAbsoluteError: 0.0846154689788818 | Loss: 0.0088662661146373 | Epoch: 81 | MeanAbsoluteE  
MeanAbsoluteError: 0.0338572300970554 | Loss: 0.0017945189230466 | Epoch: 83 | MeanAbsoluteE  
MeanAbsoluteError: 0.0377095714211464 | Loss: 0.0023885265064354 | Epoch: 85 | MeanAbsoluteE  
MeanAbsoluteError: 0.0580969750881195 | Loss: 0.0039928689724408 | Early stopping at epoch 86  
Fold: 2  
Epoch: 1 | MeanAbsoluteError: 0.1210427284240723 | Loss: 0.0232378422425917 | Epoch: 2 |  
  
MeanAbsoluteError: 0.0971136540174484 | Loss: 0.0145270234117141 | Epoch: 3 | MeanAbsoluteErr  
MeanAbsoluteError: 0.0941592678427696 | Loss: 0.0136342877880312 | Epoch: 5 | MeanAbsoluteErr  
MeanAbsoluteError: 0.0576534420251846 | Loss: 0.0057649175242449 | Epoch: 7 | MeanAbsoluteErr  
MeanAbsoluteError: 0.0551515594124794 | Loss: 0.0052624709832554 | Epoch: 9 | MeanAbsoluteErr  
MeanAbsoluteError: 0.0574899241328239 | Loss: 0.0052092311939654 | Epoch: 11 | MeanAbsoluteE  
MeanAbsoluteError: 0.0916891843080521 | Loss: 0.0108246649973668 | Epoch: 13 | MeanAbsoluteE  
MeanAbsoluteError: 0.0474723912775517 | Loss: 0.0032757236264073 | Epoch: 15 | MeanAbsoluteE  
MeanAbsoluteError: 0.0662908628582954 | Loss: 0.0076314375485079 | Epoch: 17 | MeanAbsoluteE  
MeanAbsoluteError: 0.0578889809548855 | Loss: 0.0054974844947887 | Epoch: 19 | MeanAbsoluteE  
MeanAbsoluteError: 0.0409970618784428 | Loss: 0.0031804373093809 | Epoch: 21 | MeanAbsoluteE  
MeanAbsoluteError: 0.0581042952835560 | Loss: 0.0053307732674651 | Epoch: 23 | MeanAbsoluteE
```

MeanAbsoluteError: 0.0790787190198898		Loss: 0.0077061902803297		Epoch: 25		MeanAbsoluteError: 0.0790787190198898
MeanAbsoluteError: 0.0446581616997719		Loss: 0.0027957009241128		Epoch: 27		MeanAbsoluteError: 0.0446581616997719
MeanAbsoluteError: 0.0584244057536125		Loss: 0.0042952514038636		Epoch: 29		MeanAbsoluteError: 0.0584244057536125
MeanAbsoluteError: 0.0634226948022842		Loss: 0.0053216522165502		Epoch: 31		MeanAbsoluteError: 0.0634226948022842
MeanAbsoluteError: 0.0327978432178497		Loss: 0.0021329234766129		Epoch: 33		MeanAbsoluteError: 0.0327978432178497
MeanAbsoluteError: 0.0735566541552544		Loss: 0.0066518652825975		Epoch: 35		MeanAbsoluteError: 0.0735566541552544
MeanAbsoluteError: 0.0507698543369770		Loss: 0.0044060353762828		Epoch: 37		MeanAbsoluteError: 0.0507698543369770
MeanAbsoluteError: 0.0340292416512966		Loss: 0.0019375911125770		Epoch: 39		MeanAbsoluteError: 0.0340292416512966
MeanAbsoluteError: 0.0952877774834633		Loss: 0.0101471229169804		Epoch: 41		MeanAbsoluteError: 0.0952877774834633
MeanAbsoluteError: 0.0570842325687408		Loss: 0.0045551993490125		Epoch: 43		MeanAbsoluteError: 0.0570842325687408
MeanAbsoluteError: 0.0544559583067894		Loss: 0.0043733635057624		Epoch: 45		MeanAbsoluteError: 0.0544559583067894
MeanAbsoluteError: 0.0533788762986660		Loss: 0.0041271784150292		Epoch: 47		MeanAbsoluteError: 0.0533788762986660
MeanAbsoluteError: 0.0498334281146526		Loss: 0.0033228628677674		Epoch: 49		MeanAbsoluteError: 0.0498334281146526
MeanAbsoluteError: 0.1092068180441856		Loss: 0.0138584222071446		Epoch: 51		MeanAbsoluteError: 0.1092068180441856
MeanAbsoluteError: 0.0257588960230350		Loss: 0.0013870140201806		Epoch: 53		MeanAbsoluteError: 0.0257588960230350
MeanAbsoluteError: 0.0330822132527828		Loss: 0.0016881228478339		Epoch: 55		MeanAbsoluteError: 0.0330822132527828
MeanAbsoluteError: 0.0334593765437603		Loss: 0.0021068099259327		Epoch: 57		MeanAbsoluteError: 0.0334593765437603
MeanAbsoluteError: 0.0374362431466579		Loss: 0.0028740897541866		Epoch: 59		MeanAbsoluteError: 0.0374362431466579
MeanAbsoluteError: 0.0476181395351887		Loss: 0.0032054041422760		Epoch: 61		MeanAbsoluteError: 0.0476181395351887

MeanAbsoluteError:	0.0221635792404413		Loss:	0.0015338865274456		Epoch:	63		MeanAbsoluteError:
MeanAbsoluteError:	0.0344574041664600		Loss:	0.0022858139217043		Epoch:	65		MeanAbsoluteError:
MeanAbsoluteError:	0.0327552743256092		Loss:	0.0020217526448855		Epoch:	67		MeanAbsoluteError:
MeanAbsoluteError:	0.0421554371714592		Loss:	0.0023812933986147		Epoch:	69		MeanAbsoluteError:
MeanAbsoluteError:	0.0337637700140476		Loss:	0.0027833441003727		Epoch:	71		MeanAbsoluteError:
MeanAbsoluteError:	0.0341422557830811		Loss:	0.0019972750675291		Epoch:	73		MeanAbsoluteError:
MeanAbsoluteError:	0.0273774806410074		Loss:	0.0013954009135397		Epoch:	75		MeanAbsoluteError:
MeanAbsoluteError:	0.0364593453705311		Loss:	0.0023139983129043		Epoch:	77		MeanAbsoluteError:
MeanAbsoluteError:	0.0466333962976933		Loss:	0.0025956316743619		Epoch:	79		MeanAbsoluteError:
MeanAbsoluteError:	0.0425164140760899		Loss:	0.0032064071632563		Epoch:	81		MeanAbsoluteError:
MeanAbsoluteError:	0.0778444558382034		Loss:	0.0071815219886888		Epoch:	83		MeanAbsoluteError:
MeanAbsoluteError:	0.0302984956651926		Loss:	0.0017717327886763		Epoch:	85		MeanAbsoluteError:
MeanAbsoluteError:	0.0493060834705830		Loss:	0.0034059206417833		Epoch:	87		MeanAbsoluteError:
MeanAbsoluteError:	0.0381759330630302		Loss:	0.0022709620176241		Epoch:	89		MeanAbsoluteError:
MeanAbsoluteError:	0.0505503304302692		Loss:	0.0033935281770447		Epoch:	91		MeanAbsoluteError:
MeanAbsoluteError:	0.0356516614556313		Loss:	0.0020675322963283		Epoch:	93		MeanAbsoluteError:
MeanAbsoluteError:	0.0703497678041458		Loss:	0.0053893277027573		Epoch:	95		MeanAbsoluteError:
MeanAbsoluteError:	0.0345632173120975		Loss:	0.0026033959696248		Epoch:	97		MeanAbsoluteError:
MeanAbsoluteError:	0.0295199174433947		Loss:	0.0016001898822231		Epoch:	99		MeanAbsoluteError:

MeanAbsoluteError: 0.0251525696367025		Loss: 0.0010048355347513		Epoch: 101		MeanAbsoluteError: 0.0251525696367025
MeanAbsoluteError: 0.0552912652492523		Loss: 0.0045466636164257		Epoch: 103		MeanAbsoluteError: 0.0552912652492523
MeanAbsoluteError: 0.0369178503751755		Loss: 0.0022814114748214		Epoch: 105		MeanAbsoluteError: 0.0369178503751755
MeanAbsoluteError: 0.0337572507560253		Loss: 0.0018191981138303		Epoch: 107		MeanAbsoluteError: 0.0337572507560253
Fold: 3						
Epoch: 1						
MeanAbsoluteError: 0.1106576845049858		Loss: 0.0189367327480935		Epoch: 2		MeanAbsoluteError: 0.1106576845049858
MeanAbsoluteError: 0.0982738807797432		Loss: 0.0140791730238841		Epoch: 4		MeanAbsoluteError: 0.0982738807797432
MeanAbsoluteError: 0.1125353351235390		Loss: 0.0161447196554106		Epoch: 6		MeanAbsoluteError: 0.1125353351235390
MeanAbsoluteError: 0.0799888148903847		Loss: 0.0081726278966436		Epoch: 8		MeanAbsoluteError: 0.0799888148903847
MeanAbsoluteError: 0.0800606906414032		Loss: 0.0099723924560329		Epoch: 10		MeanAbsoluteError: 0.0800606906414032
MeanAbsoluteError: 0.0613486431539059		Loss: 0.0054539138128838		Epoch: 12		MeanAbsoluteError: 0.0613486431539059
MeanAbsoluteError: 0.0422636717557907		Loss: 0.0034522012664148		Epoch: 14		MeanAbsoluteError: 0.0422636717557907
MeanAbsoluteError: 0.0377292856574059		Loss: 0.0022977507413508		Epoch: 16		MeanAbsoluteError: 0.0377292856574059
MeanAbsoluteError: 0.0457675121724606		Loss: 0.0028961827315820		Epoch: 18		MeanAbsoluteError: 0.0457675121724606
MeanAbsoluteError: 0.0386841967701912		Loss: 0.0023773076917188		Epoch: 20		MeanAbsoluteError: 0.0386841967701912
MeanAbsoluteError: 0.0304872654378414		Loss: 0.0021557314657212		Epoch: 22		MeanAbsoluteError: 0.0304872654378414
MeanAbsoluteError: 0.0281046796590090		Loss: 0.0012135672687481		Epoch: 24		MeanAbsoluteError: 0.0281046796590090
MeanAbsoluteError: 0.0270831491798162		Loss: 0.0012781705378214		Epoch: 26		MeanAbsoluteError: 0.0270831491798162
MeanAbsoluteError: 0.0480317361652851		Loss: 0.0031511885955786		Epoch: 28		MeanAbsoluteError: 0.0480317361652851

MeanAbsoluteError: 0.0747844353318214 | Loss: 0.0094273441220419 | Epoch: 6 | MeanAbsoluteError: 0.0722914189100266 | Loss: 0.0086861469138127 | Epoch: 8 | MeanAbsoluteError: 0.0745676234364510 | Loss: 0.0077011100052354 | Epoch: 10 | MeanAbsoluteError: 0.0700660869479179 | Loss: 0.0077364158530075 | Epoch: 12 | MeanAbsoluteError: 0.0842755809426308 | Loss: 0.0093502603435459 | Epoch: 14 | MeanAbsoluteError: 0.0827620476484299 | Loss: 0.0099086636462464 | Epoch: 16 | MeanAbsoluteError: 0.0719112083315849 | Loss: 0.0065859162535232 | Epoch: 18 | MeanAbsoluteError: 0.0839462280273438 | Loss: 0.0106433534708161 | Epoch: 20 | MeanAbsoluteError: 0.0494406186044216 | Loss: 0.0042909547337331 | Epoch: 22 | MeanAbsoluteError: 0.0596068389713764 | Loss: 0.0046383107815368 | Epoch: 24 | MeanAbsoluteError: 0.0434372238814831 | Loss: 0.0040412821242801 | Epoch: 26 | MeanAbsoluteError: 0.0420030318200588 | Loss: 0.0033097558099633 | Epoch: 28 | MeanAbsoluteError: 0.0713812410831451 | Loss: 0.0067926097231416 | Epoch: 30 | MeanAbsoluteError: 0.0400275103747845 | Loss: 0.0031396552356175 | Epoch: 32 | MeanAbsoluteError: 0.0456048399209976 | Loss: 0.0029108393937349 | Epoch: 34 | MeanAbsoluteError: 0.0454170741140842 | Loss: 0.0033794439111191 | Epoch: 36 | MeanAbsoluteError: 0.0590975955128670 | Loss: 0.0050150440074503 | Epoch: 38 | MeanAbsoluteError: 0.0327249653637409 | Loss: 0.0020687592572918 | Epoch: 40 | MeanAbsoluteError: 0.0450308769941330 | Loss: 0.0037826731210001 | Epoch: 42 | MeanAbsoluteError:

MeanAbsoluteError:	0.0262169465422630		Loss:	0.0015466173667497		Epoch:	44		MeanAbsoluteError:	0.0262169465422630
MeanAbsoluteError:	0.0598567016422749		Loss:	0.0047310979571193		Epoch:	46		MeanAbsoluteError:	0.0598567016422749
MeanAbsoluteError:	0.0341865122318268		Loss:	0.0020197276026011		Epoch:	48		MeanAbsoluteError:	0.0341865122318268
MeanAbsoluteError:	0.0489186793565750		Loss:	0.0039847489410581		Epoch:	50		MeanAbsoluteError:	0.0489186793565750
MeanAbsoluteError:	0.0466285236179829		Loss:	0.0036128816964964		Epoch:	52		MeanAbsoluteError:	0.0466285236179829
MeanAbsoluteError:	0.0297894421964884		Loss:	0.0019253837862589		Epoch:	54		MeanAbsoluteError:	0.0297894421964884
MeanAbsoluteError:	0.0391364395618439		Loss:	0.0027718948504816		Epoch:	56		MeanAbsoluteError:	0.0391364395618439
MeanAbsoluteError:	0.0443273447453976		Loss:	0.0029747906194713		Epoch:	58		MeanAbsoluteError:	0.0443273447453976
MeanAbsoluteError:	0.0366040430963039		Loss:	0.0023358557328510		Epoch:	60		MeanAbsoluteError:	0.0366040430963039
MeanAbsoluteError:	0.0370644293725491		Loss:	0.0034640602848063		Epoch:	62		MeanAbsoluteError:	0.0370644293725491
MeanAbsoluteError:	0.0481804646551609		Loss:	0.0034175076289102		Epoch:	64		MeanAbsoluteError:	0.0481804646551609
MeanAbsoluteError:	0.0460330620408058		Loss:	0.0036752702390703		Epoch:	66		MeanAbsoluteError:	0.0460330620408058
MeanAbsoluteError:	0.0241170693188906		Loss:	0.0014083647334841		Epoch:	68		MeanAbsoluteError:	0.0241170693188906
MeanAbsoluteError:	0.0578094273805618		Loss:	0.0043261012671372		Epoch:	70		MeanAbsoluteError:	0.0578094273805618
MeanAbsoluteError:	0.0277659706771374		Loss:	0.0014435418774016		Epoch:	72		MeanAbsoluteError:	0.0277659706771374
MeanAbsoluteError:	0.0506163984537125		Loss:	0.0042302688727012		Epoch:	74		MeanAbsoluteError:	0.0506163984537125
MeanAbsoluteError:	0.0327108353376389		Loss:	0.0022147434400932		Epoch:	76		MeanAbsoluteError:	0.0327108353376389
MeanAbsoluteError:	0.0376440733671188		Loss:	0.0027790313358240		Epoch:	78		MeanAbsoluteError:	0.0376440733671188
MeanAbsoluteError:	0.0539066120982170		Loss:	0.0045657978715518		Epoch:	80		MeanAbsoluteError:	0.0539066120982170

MeanAbsoluteError:	0.0229254085570574		Loss:	0.0015004761315560		Epoch:	82		MeanAbsoluteError:	0.0229254085570574
MeanAbsoluteError:	0.0466855205595493		Loss:	0.0039504610064726		Epoch:	84		MeanAbsoluteError:	0.0466855205595493
MeanAbsoluteError:	0.0330653153359890		Loss:	0.0019100183003152		Epoch:	86		MeanAbsoluteError:	0.0330653153359890
MeanAbsoluteError:	0.0244446750730276		Loss:	0.0012023922754452		Epoch:	88		MeanAbsoluteError:	0.0244446750730276
MeanAbsoluteError:	0.0284834727644920		Loss:	0.0016851099790074		Epoch:	90		MeanAbsoluteError:	0.0284834727644920
MeanAbsoluteError:	0.0294019300490618		Loss:	0.0018419920592211		Epoch:	92		MeanAbsoluteError:	0.0294019300490618
MeanAbsoluteError:	0.0390585213899612		Loss:	0.0023200103493694		Epoch:	94		MeanAbsoluteError:	0.0390585213899612
MeanAbsoluteError:	0.0312719047069550		Loss:	0.0025763658766384		Epoch:	96		MeanAbsoluteError:	0.0312719047069550
MeanAbsoluteError:	0.0457846671342850		Loss:	0.0032055502435049		Epoch:	98		MeanAbsoluteError:	0.0457846671342850
MeanAbsoluteError:	0.0523834042251110		Loss:	0.0040996556725496		Epoch:	100		MeanAbsoluteError:	0.0523834042251110
MeanAbsoluteError:	0.0297599583864212		Loss:	0.0015300053346436		Epoch:	102		MeanAbsoluteError:	0.0297599583864212
MeanAbsoluteError:	0.0243237614631653		Loss:	0.0014045981448502		Epoch:	104		MeanAbsoluteError:	0.0243237614631653
MeanAbsoluteError:	0.0477706827223301		Loss:	0.0033327991124959		Epoch:	106		MeanAbsoluteError:	0.0477706827223301
MeanAbsoluteError:	0.0733323544263840		Loss:	0.0068320642010524		Epoch:	108		MeanAbsoluteError:	0.0733323544263840
MeanAbsoluteError:	0.0290016178041697		Loss:	0.0013661358209972		Epoch:	110		MeanAbsoluteError:	0.0290016178041697
MeanAbsoluteError:	0.0356074534356594		Loss:	0.0025239702797710		Epoch:	112		MeanAbsoluteError:	0.0356074534356594
MeanAbsoluteError:	0.0463360361754894		Loss:	0.0040385628459402		Epoch:	114		MeanAbsoluteError:	0.0463360361754894
MeanAbsoluteError:	0.0302158910781145		Loss:	0.0027975914249859		Epoch:	116		MeanAbsoluteError:	0.0302158910781145
MeanAbsoluteError:	0.0242009498178959		Loss:	0.0009381682835878		Epoch:	118		MeanAbsoluteError:	0.0242009498178959

MeanAbsoluteError:	0.0911471396684647		Loss:	0.0121865520755259		Epoch:	7		MeanAbsoluteError:	0.0911471396684647
MeanAbsoluteError:	0.1259443312883377		Loss:	0.0199666316262805		Epoch:	9		MeanAbsoluteError:	0.1259443312883377
MeanAbsoluteError:	0.0907426178455353		Loss:	0.0101297019192806		Epoch:	11		MeanAbsoluteError:	0.0907426178455353
MeanAbsoluteError:	0.1286324560642242		Loss:	0.0217508662921878		Epoch:	13		MeanAbsoluteError:	0.1286324560642242
MeanAbsoluteError:	0.0450498312711716		Loss:	0.0035154006014077		Epoch:	15		MeanAbsoluteError:	0.0450498312711716
MeanAbsoluteError:	0.1086339652538300		Loss:	0.0142873059958220		Epoch:	17		MeanAbsoluteError:	0.1086339652538300
MeanAbsoluteError:	0.0664475932717323		Loss:	0.0057559199452114		Epoch:	19		MeanAbsoluteError:	0.0664475932717323
MeanAbsoluteError:	0.0537449233233929		Loss:	0.0045036757543970		Epoch:	21		MeanAbsoluteError:	0.0537449233233929
MeanAbsoluteError:	0.1098860353231430		Loss:	0.0169710179146093		Epoch:	23		MeanAbsoluteError:	0.1098860353231430
MeanAbsoluteError:	0.0625247731804848		Loss:	0.0053505155419071		Epoch:	25		MeanAbsoluteError:	0.0625247731804848
MeanAbsoluteError:	0.0708045139908791		Loss:	0.0078202823284440		Epoch:	27		MeanAbsoluteError:	0.0708045139908791
MeanAbsoluteError:	0.0740147307515144		Loss:	0.0073594601753239		Epoch:	29		MeanAbsoluteError:	0.0740147307515144
MeanAbsoluteError:	0.0415130034089088		Loss:	0.0035930987528095		Epoch:	31		MeanAbsoluteError:	0.0415130034089088
MeanAbsoluteError:	0.0427530556917191		Loss:	0.0042804747533340		Epoch:	33		MeanAbsoluteError:	0.0427530556917191
MeanAbsoluteError:	0.0303096286952496		Loss:	0.0016219085037637		Epoch:	35		MeanAbsoluteError:	0.0303096286952496
MeanAbsoluteError:	0.0342826209962368		Loss:	0.0020417858434554		Epoch:	37		MeanAbsoluteError:	0.0342826209962368
MeanAbsoluteError:	0.0819529369473457		Loss:	0.0085897359352272		Epoch:	39		MeanAbsoluteError:	0.0819529369473457
MeanAbsoluteError:	0.0411276705563068		Loss:	0.0033256742798795		Epoch:	41		MeanAbsoluteError:	0.0411276705563068
MeanAbsoluteError:	0.0454586856067181		Loss:	0.0035414176348310		Epoch:	43		MeanAbsoluteError:	0.0454586856067181

MeanAbsoluteError:	0.0417152270674706		Loss:	0.0031107160388134		Epoch:	45		MeanAbsoluteError:	0.0417152270674706
MeanAbsoluteError:	0.0276019889861345		Loss:	0.0021961276988105		Epoch:	47		MeanAbsoluteError:	0.0276019889861345
MeanAbsoluteError:	0.0443143174052238		Loss:	0.0029611022910103		Epoch:	49		MeanAbsoluteError:	0.0443143174052238
MeanAbsoluteError:	0.0462977215647697		Loss:	0.0032892203442036		Epoch:	51		MeanAbsoluteError:	0.0462977215647697
MeanAbsoluteError:	0.0283489897847176		Loss:	0.0016895886040472		Epoch:	53		MeanAbsoluteError:	0.0283489897847176
MeanAbsoluteError:	0.0611473359167576		Loss:	0.0051516056992114		Epoch:	55		MeanAbsoluteError:	0.0611473359167576
MeanAbsoluteError:	0.0874474719166756		Loss:	0.0087892629134540		Epoch:	57		MeanAbsoluteError:	0.0874474719166756
MeanAbsoluteError:	0.0342605821788311		Loss:	0.0027194319028730		Epoch:	59		MeanAbsoluteError:	0.0342605821788311
MeanAbsoluteError:	0.0257919151335955		Loss:	0.0012255321735910		Epoch:	61		MeanAbsoluteError:	0.0257919151335955
MeanAbsoluteError:	0.0615654364228249		Loss:	0.0045509674013234		Epoch:	63		MeanAbsoluteError:	0.0615654364228249
MeanAbsoluteError:	0.0449857600033283		Loss:	0.0041857361659193		Epoch:	65		MeanAbsoluteError:	0.0449857600033283
MeanAbsoluteError:	0.0266420654952526		Loss:	0.0012135091954126		Epoch:	67		MeanAbsoluteError:	0.0266420654952526
MeanAbsoluteError:	0.0688930079340935		Loss:	0.0073656662761305		Epoch:	69		MeanAbsoluteError:	0.0688930079340935
MeanAbsoluteError:	0.0329585596919060		Loss:	0.0021188824679344		Epoch:	71		MeanAbsoluteError:	0.0329585596919060
MeanAbsoluteError:	0.0498217679560184		Loss:	0.0039318983371441		Epoch:	73		MeanAbsoluteError:	0.0498217679560184
MeanAbsoluteError:	0.0312236174941063		Loss:	0.0022571678318155		Epoch:	75		MeanAbsoluteError:	0.0312236174941063
MeanAbsoluteError:	0.0359724126756191		Loss:	0.0028056018726112		Epoch:	77		MeanAbsoluteError:	0.0359724126756191
MeanAbsoluteError:	0.0335180014371872		Loss:	0.0030893175146328		Epoch:	79		MeanAbsoluteError:	0.0335180014371872
MeanAbsoluteError:	0.0293328072875738		Loss:	0.0018226521105344		Epoch:	81		MeanAbsoluteError:	0.0293328072875738

MeanAbsoluteError:	0.0255958512425423		Loss:	0.0021173147621224		Epoch:	83		MeanAbsoluteError:	0.0255958512425423
MeanAbsoluteError:	0.0278031583875418		Loss:	0.0014556702002525		Epoch:	85		MeanAbsoluteError:	0.0278031583875418
MeanAbsoluteError:	0.0231483560055494		Loss:	0.0013799175749927		Epoch:	87		MeanAbsoluteError:	0.0231483560055494
MeanAbsoluteError:	0.0420096628367901		Loss:	0.0028526257105673		Epoch:	89		MeanAbsoluteError:	0.0420096628367901
MeanAbsoluteError:	0.0194805413484573		Loss:	0.0006444030486119		Epoch:	91		MeanAbsoluteError:	0.0194805413484573
MeanAbsoluteError:	0.0275841187685728		Loss:	0.0016856150446424		Epoch:	93		MeanAbsoluteError:	0.0275841187685728
MeanAbsoluteError:	0.0282230041921139		Loss:	0.0014793445365145		Epoch:	95		MeanAbsoluteError:	0.0282230041921139
MeanAbsoluteError:	0.0555804073810577		Loss:	0.0041483210292287		Epoch:	97		MeanAbsoluteError:	0.0555804073810577
MeanAbsoluteError:	0.0342480652034283		Loss:	0.0020118495158385		Epoch:	99		MeanAbsoluteError:	0.0342480652034283
MeanAbsoluteError:	0.0288295391947031		Loss:	0.0014184577493534		Epoch:	101		MeanAbsoluteError:	0.0288295391947031
MeanAbsoluteError:	0.0339241400361061		Loss:	0.0029046865798031		Epoch:	103		MeanAbsoluteError:	0.0339241400361061
MeanAbsoluteError:	0.0508102923631668		Loss:	0.0035383596729774		Epoch:	105		MeanAbsoluteError:	0.0508102923631668
MeanAbsoluteError:	0.0505762770771980		Loss:	0.0044018738509084		Epoch:	107		MeanAbsoluteError:	0.0505762770771980
MeanAbsoluteError:	0.0291441138833761		Loss:	0.0015848892149874		Epoch:	109		MeanAbsoluteError:	0.0291441138833761
MeanAbsoluteError:	0.0226606409996748		Loss:	0.0009156773851898		Epoch:	111		MeanAbsoluteError:	0.0226606409996748
MeanAbsoluteError:	0.0711091011762619		Loss:	0.0067013068876874		Epoch:	113		MeanAbsoluteError:	0.0711091011762619
MeanAbsoluteError:	0.0554409921169281		Loss:	0.0040316201316623		Epoch:	115		MeanAbsoluteError:	0.0554409921169281
MeanAbsoluteError:	0.0249174181371927		Loss:	0.0009794597773669		Epoch:	117		MeanAbsoluteError:	0.0249174181371927
MeanAbsoluteError:	0.0242767669260502		Loss:	0.0012442641555726		Epoch:	119		MeanAbsoluteError:	0.0242767669260502

MeanAbsoluteError:	0.0966121181845665		Loss:	0.0127073350147559		Epoch:	35		MeanAbsoluteError:	0.0966121181845665
MeanAbsoluteError:	0.0628375560045242		Loss:	0.0059970660905282		Epoch:	37		MeanAbsoluteError:	0.0628375560045242
MeanAbsoluteError:	0.0451613143086433		Loss:	0.0030788400461181		Epoch:	39		MeanAbsoluteError:	0.0451613143086433
MeanAbsoluteError:	0.0455550663173199		Loss:	0.0032295483845071		Epoch:	41		MeanAbsoluteError:	0.0455550663173199
MeanAbsoluteError:	0.0602814070880413		Loss:	0.0056687516625971		Epoch:	43		MeanAbsoluteError:	0.0602814070880413
MeanAbsoluteError:	0.0426491312682629		Loss:	0.0029243143239560		Epoch:	45		MeanAbsoluteError:	0.0426491312682629
MeanAbsoluteError:	0.0582808442413807		Loss:	0.0042237151915637		Epoch:	47		MeanAbsoluteError:	0.0582808442413807
MeanAbsoluteError:	0.0433768704533577		Loss:	0.0034941920491222		Epoch:	49		MeanAbsoluteError:	0.0433768704533577
MeanAbsoluteError:	0.0419469252228737		Loss:	0.0027373476500193		Epoch:	51		MeanAbsoluteError:	0.0419469252228737
MeanAbsoluteError:	0.0795767754316330		Loss:	0.0082582135040026		Epoch:	53		MeanAbsoluteError:	0.0795767754316330
MeanAbsoluteError:	0.0735091120004654		Loss:	0.0075749174440996		Epoch:	55		MeanAbsoluteError:	0.0735091120004654
MeanAbsoluteError:	0.0441880561411381		Loss:	0.0034758094698191		Epoch:	57		MeanAbsoluteError:	0.0441880561411381
MeanAbsoluteError:	0.0750773921608925		Loss:	0.0064214364840434		Epoch:	59		MeanAbsoluteError:	0.0750773921608925
MeanAbsoluteError:	0.0521817691624165		Loss:	0.0050249007136489		Epoch:	61		MeanAbsoluteError:	0.0521817691624165
MeanAbsoluteError:	0.0561900287866592		Loss:	0.0050264165306894		Epoch:	63		MeanAbsoluteError:	0.0561900287866592
MeanAbsoluteError:	0.0467034578323364		Loss:	0.0033031822218058		Epoch:	65		MeanAbsoluteError:	0.0467034578323364
MeanAbsoluteError:	0.0334020406007767		Loss:	0.0020290773525351		Epoch:	67		MeanAbsoluteError:	0.0334020406007767
MeanAbsoluteError:	0.0319847911596298		Loss:	0.0021565292767124		Epoch:	69		MeanAbsoluteError:	0.0319847911596298
MeanAbsoluteError:	0.0428840592503548		Loss:	0.0029401911865884		Epoch:	71		MeanAbsoluteError:	0.0428840592503548

MeanAbsoluteError:	0.0272354967892170		Loss:	0.0015840791871037		Epoch:	73		MeanAbsoluteError:	0.0272354967892170
MeanAbsoluteError:	0.0336233936250210		Loss:	0.0019344711024762		Epoch:	75		MeanAbsoluteError:	0.0336233936250210
MeanAbsoluteError:	0.0311421751976013		Loss:	0.0017377148391321		Epoch:	77		MeanAbsoluteError:	0.0311421751976013
MeanAbsoluteError:	0.0717094242572784		Loss:	0.0061883135304715		Epoch:	79		MeanAbsoluteError:	0.0717094242572784
MeanAbsoluteError:	0.0873696133494377		Loss:	0.0100450945946460		Epoch:	81		MeanAbsoluteError:	0.0873696133494377
MeanAbsoluteError:	0.0337243117392063		Loss:	0.0020317407839824		Epoch:	83		MeanAbsoluteError:	0.0337243117392063
MeanAbsoluteError:	0.0357621125876904		Loss:	0.0020573903983369		Epoch:	85		MeanAbsoluteError:	0.0357621125876904
Fold: 7										
Epoch: 1										
MeanAbsoluteError:	0.1181101053953171		Loss:	0.0207998535524194		Epoch:	2		MeanAbsoluteError:	0.1181101053953171
MeanAbsoluteError:	0.1253294646739960		Loss:	0.0225796740358839		Epoch:	4		MeanAbsoluteError:	0.1253294646739960
MeanAbsoluteError:	0.0905140936374664		Loss:	0.0132385202182027		Epoch:	6		MeanAbsoluteError:	0.0905140936374664
MeanAbsoluteError:	0.0688776522874832		Loss:	0.0072345151028668		Epoch:	8		MeanAbsoluteError:	0.0688776522874832
MeanAbsoluteError:	0.0736842602491379		Loss:	0.0072026569933559		Epoch:	10		MeanAbsoluteError:	0.0736842602491379
MeanAbsoluteError:	0.0782508105039597		Loss:	0.0082479237507169		Epoch:	12		MeanAbsoluteError:	0.0782508105039597
MeanAbsoluteError:	0.0621398575603962		Loss:	0.0054982596148665		Epoch:	14		MeanAbsoluteError:	0.0621398575603962
MeanAbsoluteError:	0.0335745848715305		Loss:	0.0021558927384411		Epoch:	16		MeanAbsoluteError:	0.0335745848715305
MeanAbsoluteError:	0.0458005107939243		Loss:	0.0042439127484193		Epoch:	18		MeanAbsoluteError:	0.0458005107939243
MeanAbsoluteError:	0.0441776514053345		Loss:	0.0027292787921257		Epoch:	20		MeanAbsoluteError:	0.0441776514053345
MeanAbsoluteError:	0.0359511263668537		Loss:	0.0021718123282951		Epoch:	22		MeanAbsoluteError:	0.0359511263668537

MeanAbsoluteError:	0.0367610566318035		Loss:	0.0028986775992402		Epoch:	24		MeanAbsoluteError:	0.0367610566318035
MeanAbsoluteError:	0.0441513620316982		Loss:	0.0028870862711651		Epoch:	26		MeanAbsoluteError:	0.0441513620316982
MeanAbsoluteError:	0.0543140992522240		Loss:	0.0040151120091860		Epoch:	28		MeanAbsoluteError:	0.0543140992522240
MeanAbsoluteError:	0.0288309101015329		Loss:	0.0014520277461718		Epoch:	30		MeanAbsoluteError:	0.0288309101015329
MeanAbsoluteError:	0.0259126964956522		Loss:	0.0012397431978920		Epoch:	32		MeanAbsoluteError:	0.0259126964956522
MeanAbsoluteError:	0.0317005664110184		Loss:	0.0014227875452399		Epoch:	34		MeanAbsoluteError:	0.0317005664110184
MeanAbsoluteError:	0.0665730759501457		Loss:	0.0057888871703583		Epoch:	36		MeanAbsoluteError:	0.0665730759501457
MeanAbsoluteError:	0.0275910180062056		Loss:	0.0014056670389926		Epoch:	38		MeanAbsoluteError:	0.0275910180062056
MeanAbsoluteError:	0.0249668676406145		Loss:	0.0011471343190911		Epoch:	40		MeanAbsoluteError:	0.0249668676406145
MeanAbsoluteError:	0.0731166526675224		Loss:	0.0063893144878630		Epoch:	42		MeanAbsoluteError:	0.0731166526675224
MeanAbsoluteError:	0.0641200914978981		Loss:	0.0059991085257095		Epoch:	44		MeanAbsoluteError:	0.0641200914978981
MeanAbsoluteError:	0.0999278351664543		Loss:	0.0135999687660772		Epoch:	46		MeanAbsoluteError:	0.0999278351664543
MeanAbsoluteError:	0.0250954087823629		Loss:	0.0011521725956789		Epoch:	48		MeanAbsoluteError:	0.0250954087823629
MeanAbsoluteError:	0.0545343905687332		Loss:	0.0040793491151327		Epoch:	50		MeanAbsoluteError:	0.0545343905687332
MeanAbsoluteError:	0.0466722473502159		Loss:	0.0031522543151648		Epoch:	52		MeanAbsoluteError:	0.0466722473502159
MeanAbsoluteError:	0.0592239722609520		Loss:	0.0044626619648905		Epoch:	54		MeanAbsoluteError:	0.0592239722609520
MeanAbsoluteError:	0.0551084093749523		Loss:	0.0043240901476775		Epoch:	56		MeanAbsoluteError:	0.0551084093749523
MeanAbsoluteError:	0.0525187142193317		Loss:	0.0034824634382788		Epoch:	58		MeanAbsoluteError:	0.0525187142193317
MeanAbsoluteError:	0.0404592081904411		Loss:	0.0035595883516810		Epoch:	60		MeanAbsoluteError:	0.0404592081904411

MeanAbsoluteError:	0.1431737393140793		Loss:	0.0297155630273315		Epoch:	2		MeanAbsoluteError:	0.1431737393140793
MeanAbsoluteError:	0.1129568964242935		Loss:	0.0186832501696279		Epoch:	4		MeanAbsoluteError:	0.1129568964242935
MeanAbsoluteError:	0.0553855448961258		Loss:	0.0045964737297394		Epoch:	6		MeanAbsoluteError:	0.0553855448961258
MeanAbsoluteError:	0.0534265078604221		Loss:	0.0055840972345322		Epoch:	8		MeanAbsoluteError:	0.0534265078604221
MeanAbsoluteError:	0.0946155488491058		Loss:	0.0114883354936655		Epoch:	10		MeanAbsoluteError:	0.0946155488491058
MeanAbsoluteError:	0.0351461879909039		Loss:	0.0020505198761104		Epoch:	12		MeanAbsoluteError:	0.0351461879909039
MeanAbsoluteError:	0.0439841262996197		Loss:	0.0038868219573767		Epoch:	14		MeanAbsoluteError:	0.0439841262996197
MeanAbsoluteError:	0.1192435994744301		Loss:	0.0159111410522690		Epoch:	16		MeanAbsoluteError:	0.1192435994744301
MeanAbsoluteError:	0.0729183182120323		Loss:	0.0082608756358520		Epoch:	18		MeanAbsoluteError:	0.0729183182120323
MeanAbsoluteError:	0.0503072515130043		Loss:	0.0031655384585834		Epoch:	20		MeanAbsoluteError:	0.0503072515130043
MeanAbsoluteError:	0.0945181250572205		Loss:	0.0103931006115790		Epoch:	22		MeanAbsoluteError:	0.0945181250572205
MeanAbsoluteError:	0.1143421381711960		Loss:	0.0153395225867056		Epoch:	24		MeanAbsoluteError:	0.1143421381711960
MeanAbsoluteError:	0.0397533327341080		Loss:	0.0027962174857608		Epoch:	26		MeanAbsoluteError:	0.0397533327341080
MeanAbsoluteError:	0.0651302039623260		Loss:	0.0055486265247544		Epoch:	28		MeanAbsoluteError:	0.0651302039623260
MeanAbsoluteError:	0.0416619293391705		Loss:	0.0028471509173799		Epoch:	30		MeanAbsoluteError:	0.0416619293391705
MeanAbsoluteError:	0.0665900632739067		Loss:	0.0057332516970256		Epoch:	32		MeanAbsoluteError:	0.0665900632739067
MeanAbsoluteError:	0.0287870652973652		Loss:	0.0014105396201977		Epoch:	34		MeanAbsoluteError:	0.0287870652973652
MeanAbsoluteError:	0.0240901187062263		Loss:	0.0012200706024977		Epoch:	36		MeanAbsoluteError:	0.0240901187062263
MeanAbsoluteError:	0.0459063164889812		Loss:	0.0026319806840127		Epoch:	38		MeanAbsoluteError:	0.0459063164889812

```
MeanAbsoluteError: 0.0221337098628283 | Loss: 0.0010073897898949 | Epoch: 40 | MeanAbsoluteE
MeanAbsoluteError: 0.0456602275371552 | Loss: 0.0031009974638717 | Epoch: 42 | MeanAbsoluteE
MeanAbsoluteError: 0.0592089258134365 | Loss: 0.0048350690228220 | Epoch: 44 | MeanAbsoluteE
MeanAbsoluteError: 0.0305355302989483 | Loss: 0.0014669969274949 | Epoch: 46 | MeanAbsoluteE
MeanAbsoluteError: 0.0365908853709698 | Loss: 0.0018272739700758 | Epoch: 48 | MeanAbsoluteE
MeanAbsoluteError: 0.0324256122112274 | Loss: 0.0017734024485645 | Epoch: 50 | MeanAbsoluteE
MeanAbsoluteError: 0.0339218154549599 | Loss: 0.0018092689570040 | Epoch: 52 | MeanAbsoluteE
MeanAbsoluteError: 0.0467526912689209 | Loss: 0.0033993757007500 | Epoch: 54 | MeanAbsoluteE
MeanAbsoluteError: 0.0267228260636330 | Loss: 0.0012468975417925 | Epoch: 56 | MeanAbsoluteE
MeanAbsoluteError: 0.0554147996008396 | Loss: 0.0036517197731882 | Epoch: 58 | MeanAbsoluteE
MeanAbsoluteError: 0.0930708870291710 | Loss: 0.0117698269896209 | Epoch: 60 | MeanAbsoluteE
MeanAbsoluteError: 0.0537774898111820 | Loss: 0.0038655340814820 | Epoch: 62 | MeanAbsoluteE
MeanAbsoluteError: 0.0589980781078339 | Loss: 0.0045726656501826 | Epoch: 64 | MeanAbsoluteE
MeanAbsoluteError: 0.0501065701246262 | Loss: 0.0042323238292458 | Epoch: 66 | MeanAbsoluteE
MeanAbsoluteError: 0.0429092869162560 | Loss: 0.0024519963548161 | Epoch: 68 | MeanAbsoluteE
MeanAbsoluteError: 0.0951257795095444 | Loss: 0.0117127123073890 | Epoch: 70 | MeanAbsoluteE
MeanAbsoluteError: 0.0720366686582565 | Loss: 0.0061199199408293 | Early stopping at epoch 70
Fold: 9
Epoch: 1 | MeanAbsoluteError: 0.1086572632193565 | Loss: 0.0201941847156447 | Epoch: 2 |
MeanAbsoluteError: 0.0934930890798569 | Loss: 0.0132137790967066 | Epoch: 3 | MeanAbsoluteE
```


MeanAbsoluteError:	0.0622712858021259		Loss:	0.0048845905201653		Epoch:	43		MeanAbsoluteError:
MeanAbsoluteError:	0.0280416067689657		Loss:	0.0012979594545099		Epoch:	45		MeanAbsoluteError:
MeanAbsoluteError:	0.0370311625301838		Loss:	0.0021517566817168		Epoch:	47		MeanAbsoluteError:
MeanAbsoluteError:	0.0498456619679928		Loss:	0.0033377350415461		Epoch:	49		MeanAbsoluteError:
MeanAbsoluteError:	0.0342269428074360		Loss:	0.0024432269274257		Epoch:	51		MeanAbsoluteError:
MeanAbsoluteError:	0.0275215916335583		Loss:	0.0017103685397440		Epoch:	53		MeanAbsoluteError:
MeanAbsoluteError:	0.0475216731429100		Loss:	0.0033263715449721		Epoch:	55		MeanAbsoluteError:
MeanAbsoluteError:	0.0269825179129839		Loss:	0.0015262023214466		Epoch:	57		MeanAbsoluteError:
MeanAbsoluteError:	0.0481035858392715		Loss:	0.0030576877492981		Epoch:	59		MeanAbsoluteError:
MeanAbsoluteError:	0.0348006188869476		Loss:	0.0021571580168361		Epoch:	61		MeanAbsoluteError:
MeanAbsoluteError:	0.0433100834488869		Loss:	0.0030014084320730		Epoch:	63		MeanAbsoluteError:
MeanAbsoluteError:	0.0368292331695557		Loss:	0.0022006317314943		Epoch:	65		MeanAbsoluteError:
MeanAbsoluteError:	0.0346639454364777		Loss:	0.0026473766237569		Epoch:	67		MeanAbsoluteError:
MeanAbsoluteError:	0.0393413081765175		Loss:	0.0024064552674715		Epoch:	69		MeanAbsoluteError:
MeanAbsoluteError:	0.0429924204945564		Loss:	0.0036360692734329		Epoch:	71		MeanAbsoluteError:
MeanAbsoluteError:	0.0283342413604259		Loss:	0.0014326131987708		Epoch:	73		MeanAbsoluteError:
MeanAbsoluteError:	0.0248284824192524		Loss:	0.0011423179440988		Epoch:	75		MeanAbsoluteError:
MeanAbsoluteError:	0.0541838854551315		Loss:	0.0036633575979907		Epoch:	77		MeanAbsoluteError:
MeanAbsoluteError:	0.0337822660803795		Loss:	0.0022683073050128		Epoch:	79		MeanAbsoluteError:

MeanAbsoluteError:	0.0424452647566795		Loss:	0.0025676231777582		Epoch:	81		MeanAbsoluteError:
MeanAbsoluteError:	0.0406265556812286		Loss:	0.0028339416290132		Epoch:	83		MeanAbsoluteError:
MeanAbsoluteError:	0.0894842147827148		Loss:	0.0091304753620464		Epoch:	85		MeanAbsoluteError:
MeanAbsoluteError:	0.0290435925126076		Loss:	0.0014850749284960		Epoch:	87		MeanAbsoluteError:
MeanAbsoluteError:	0.0618923716247082		Loss:	0.0047182171748808		Epoch:	89		MeanAbsoluteError:
MeanAbsoluteError:	0.0302210804075003		Loss:	0.0014596949868764		Epoch:	91		MeanAbsoluteError:
MeanAbsoluteError:	0.0830024927854538		Loss:	0.0094769958478327		Epoch:	93		MeanAbsoluteError:
MeanAbsoluteError:	0.0241335630416870		Loss:	0.0013569568803247		Epoch:	95		MeanAbsoluteError:
MeanAbsoluteError:	0.0336342975497246		Loss:	0.0016523559640448		Epoch:	97		MeanAbsoluteError:
MeanAbsoluteError:	0.0257233269512653		Loss:	0.0010881706100638		Epoch:	99		MeanAbsoluteError:
MeanAbsoluteError:	0.0739192664623260		Loss:	0.0075177455571695		Epoch:	101		MeanAbsoluteError:
MeanAbsoluteError:	0.0277944896370173		Loss:	0.0014599524751807		Epoch:	103		MeanAbsoluteError:
MeanAbsoluteError:	0.0678363814949989		Loss:	0.0059429356971612		Epoch:	105		MeanAbsoluteError:
MeanAbsoluteError:	0.0383042171597481		Loss:	0.0021507579412383		Epoch:	107		MeanAbsoluteError:
MeanAbsoluteError:	0.0220116600394249		Loss:	0.0009775411912974		Epoch:	109		MeanAbsoluteError:
MeanAbsoluteError:	0.0365606099367142		Loss:	0.0032283642839712		Epoch:	111		MeanAbsoluteError:
MeanAbsoluteError:	0.0753240957856178		Loss:	0.0066646417387976		Epoch:	113		MeanAbsoluteError:
MeanAbsoluteError:	0.0637192800641060		Loss:	0.0065724830268524		Epoch:	115		MeanAbsoluteError:
MeanAbsoluteError:	0.0314157530665398		Loss:	0.0017951420382955		Epoch:	117		MeanAbsoluteError:

MeanAbsoluteError:	0.0851664245128632		Loss:	0.0089700542605267		Epoch:	119		MeanAbsoluteError:
MeanAbsoluteError:	0.0247977878898382		Loss:	0.0009182052319654		Epoch:	121		MeanAbsoluteError:
MeanAbsoluteError:	0.0235399100929499		Loss:	0.0013098884978367		Epoch:	123		MeanAbsoluteError:
MeanAbsoluteError:	0.0518441386520863		Loss:	0.0032232807985005		Epoch:	125		MeanAbsoluteError:
MeanAbsoluteError:	0.0464971512556076		Loss:	0.0034786808559707		Epoch:	127		MeanAbsoluteError:
MeanAbsoluteError:	0.0365378223359585		Loss:	0.0024416067261392		Epoch:	129		MeanAbsoluteError:
MeanAbsoluteError:	0.0659212321043015		Loss:	0.0057641606455526		Epoch:	131		MeanAbsoluteError:
MeanAbsoluteError:	0.0267138648778200		Loss:	0.0011322076562255		Epoch:	133		MeanAbsoluteError:
MeanAbsoluteError:	0.0194982569664717		Loss:	0.0008207166695054		Epoch:	135		MeanAbsoluteError:
MeanAbsoluteError:	0.0203929059207439		Loss:	0.0008401474232624		Epoch:	137		MeanAbsoluteError:
MeanAbsoluteError:	0.0290005300194025		Loss:	0.0020525477593765		Epoch:	139		MeanAbsoluteError:
MeanAbsoluteError:	0.0212643630802631		Loss:	0.0009807098764353		Epoch:	141		MeanAbsoluteError:
MeanAbsoluteError:	0.0380935892462730		Loss:	0.0030575540373460		Epoch:	143		MeanAbsoluteError:
MeanAbsoluteError:	0.0246555376797915		Loss:	0.0012899119935285		Epoch:	145		MeanAbsoluteError:
MeanAbsoluteError:	0.0267183519899845		Loss:	0.0010897133929225		Epoch:	147		MeanAbsoluteError:
MeanAbsoluteError:	0.0276903063058853		Loss:	0.0013129490947064		Epoch:	149		MeanAbsoluteError:
MeanAbsoluteError:	0.0238710194826126		Loss:	0.0012850386410719		Epoch:	151		MeanAbsoluteError:
MeanAbsoluteError:	0.0294004250317812		Loss:	0.0022103867309992		Epoch:	153		MeanAbsoluteError:

MeanAbsoluteError:	0.0394270606338978		Loss:	0.0027102856547572		Epoch:	36		MeanAbsoluteError:	0.0394270606338978
MeanAbsoluteError:	0.0363821312785149		Loss:	0.0022981364074020		Epoch:	38		MeanAbsoluteError:	0.0363821312785149
MeanAbsoluteError:	0.0532912127673626		Loss:	0.0047393800296749		Epoch:	40		MeanAbsoluteError:	0.0532912127673626
MeanAbsoluteError:	0.0214278642088175		Loss:	0.0011383185371345		Epoch:	42		MeanAbsoluteError:	0.0214278642088175
MeanAbsoluteError:	0.0313934646546841		Loss:	0.0020190687814298		Epoch:	44		MeanAbsoluteError:	0.0313934646546841
MeanAbsoluteError:	0.0743075385689735		Loss:	0.0081483257862811		Epoch:	46		MeanAbsoluteError:	0.0743075385689735
MeanAbsoluteError:	0.0981889143586159		Loss:	0.0110198168370586		Epoch:	48		MeanAbsoluteError:	0.0981889143586159
MeanAbsoluteError:	0.0811061263084412		Loss:	0.0075183138251305		Epoch:	50		MeanAbsoluteError:	0.0811061263084412
MeanAbsoluteError:	0.0375275202095509		Loss:	0.0021379338084863		Epoch:	52		MeanAbsoluteError:	0.0375275202095509
MeanAbsoluteError:	0.0990672484040260		Loss:	0.0117675217871483		Epoch:	54		MeanAbsoluteError:	0.0990672484040260
MeanAbsoluteError:	0.0561995692551136		Loss:	0.0047397884728315		Epoch:	56		MeanAbsoluteError:	0.0561995692551136
MeanAbsoluteError:	0.0282295607030392		Loss:	0.0016251633194490		Epoch:	58		MeanAbsoluteError:	0.0282295607030392
MeanAbsoluteError:	0.0829904749989510		Loss:	0.0086207497291840		Epoch:	60		MeanAbsoluteError:	0.0829904749989510
MeanAbsoluteError:	0.0556950345635414		Loss:	0.0039507038533114		Epoch:	62		MeanAbsoluteError:	0.0556950345635414
MeanAbsoluteError:	0.0315789058804512		Loss:	0.0016308134744087		Epoch:	64		MeanAbsoluteError:	0.0315789058804512
MeanAbsoluteError:	0.0536704771220684		Loss:	0.0042195228429941		Epoch:	66		MeanAbsoluteError:	0.0536704771220684
MeanAbsoluteError:	0.0705133751034737		Loss:	0.0060252818064048		Epoch:	68		MeanAbsoluteError:	0.0705133751034737
MeanAbsoluteError:	0.0287140551954508		Loss:	0.0021043367951979		Epoch:	70		MeanAbsoluteError:	0.0287140551954508
MeanAbsoluteError:	0.0285404343158007		Loss:	0.0019902293480789		Epoch:	72		MeanAbsoluteError:	0.0285404343158007

MeanAbsoluteError:	0.0591405294835567		Loss:	0.0053762215941858		Epoch:	74		MeanAbsoluteError:	0.0591405294835567
MeanAbsoluteError:	0.0273822695016861		Loss:	0.0019253477624331		Epoch:	76		MeanAbsoluteError:	0.0273822695016861
MeanAbsoluteError:	0.0327594466507435		Loss:	0.0016105747065292		Epoch:	78		MeanAbsoluteError:	0.0327594466507435
MeanAbsoluteError:	0.0355377048254013		Loss:	0.0020530053432314		Epoch:	80		MeanAbsoluteError:	0.0355377048254013
MeanAbsoluteError:	0.0298162456601858		Loss:	0.0019593062147928		Epoch:	82		MeanAbsoluteError:	0.0298162456601858
MeanAbsoluteError:	0.0273981206119061		Loss:	0.0022060886918692		Epoch:	84		MeanAbsoluteError:	0.0273981206119061
MeanAbsoluteError:	0.0415325723588467		Loss:	0.0027466045710473		Epoch:	86		MeanAbsoluteError:	0.0415325723588467
MeanAbsoluteError:	0.0498743727803230		Loss:	0.0033780845920913		Epoch:	88		MeanAbsoluteError:	0.0498743727803230
MeanAbsoluteError:	0.0505493879318237		Loss:	0.0037510936829047		Epoch:	90		MeanAbsoluteError:	0.0505493879318237
MeanAbsoluteError:	0.0594000481069088		Loss:	0.0057154797250405		Epoch:	92		MeanAbsoluteError:	0.0594000481069088
MeanAbsoluteError:	0.0266177486628294		Loss:	0.0010481001092837		Epoch:	94		MeanAbsoluteError:	0.0266177486628294
MeanAbsoluteError:	0.0721541568636894		Loss:	0.0082577932626009		Epoch:	96		MeanAbsoluteError:	0.0721541568636894
MeanAbsoluteError:	0.0319408662617207		Loss:	0.0016206096266755		Epoch:	98		MeanAbsoluteError:	0.0319408662617207
MeanAbsoluteError:	0.0378547757863998		Loss:	0.0019473610714508		Epoch:	100		MeanAbsoluteError:	0.0378547757863998
MeanAbsoluteError:	0.0233147051185369		Loss:	0.0009550361383635		Epoch:	102		MeanAbsoluteError:	0.0233147051185369
MeanAbsoluteError:	0.0690123811364174		Loss:	0.0060489769272793		Epoch:	104		MeanAbsoluteError:	0.0690123811364174
MeanAbsoluteError:	0.0497683100402355		Loss:	0.0040739173463617		Epoch:	106		MeanAbsoluteError:	0.0497683100402355
MeanAbsoluteError:	0.0403967760503292		Loss:	0.0028004142939328		Epoch:	108		MeanAbsoluteError:	0.0403967760503292
MeanAbsoluteError:	0.0341406017541885		Loss:	0.0019116246481784		Epoch:	110		MeanAbsoluteError:	0.0341406017541885

19.10.4 Detailed Hyperparameter Plots

```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

```
l1: 27.625846168352734
dropout_prob: 1.7781767444973497
patience: 100.0
sgd_momentum: 100.0
```

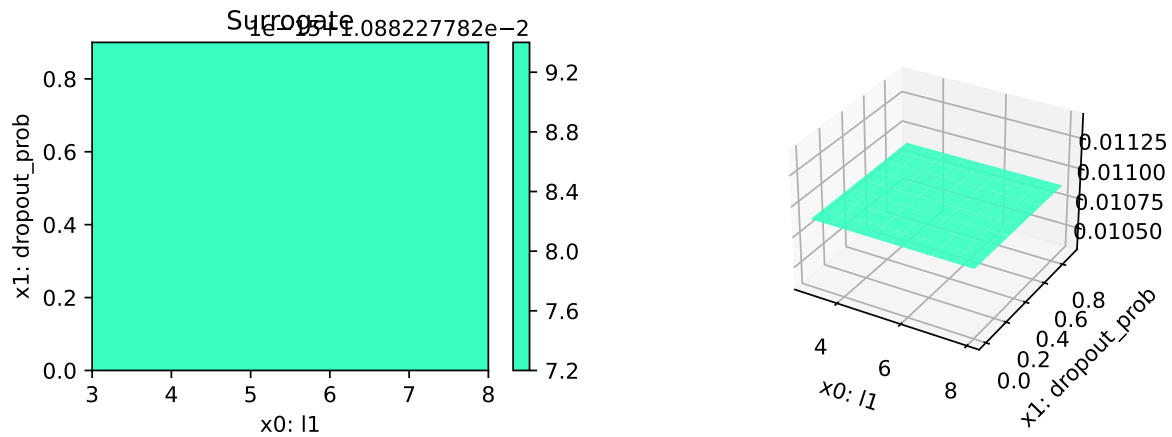
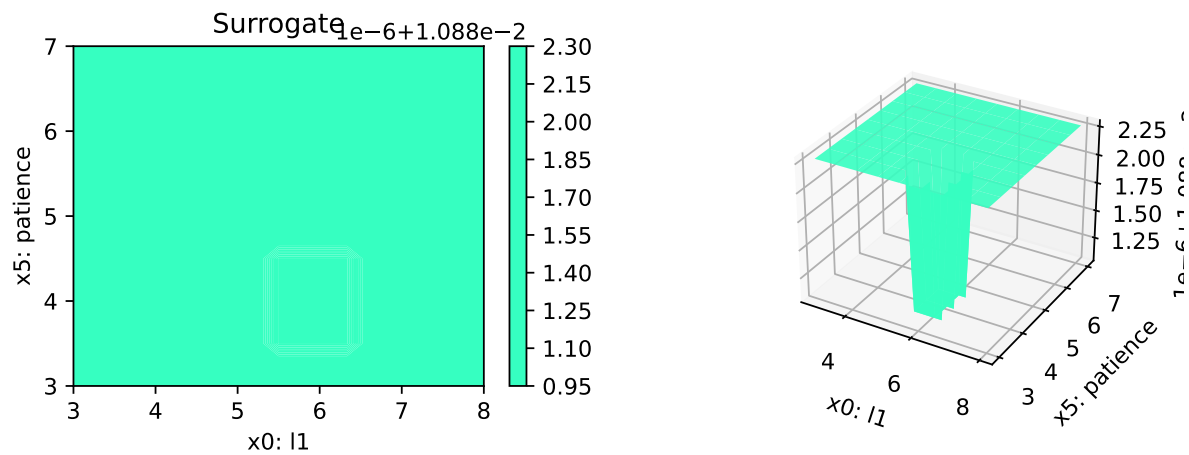
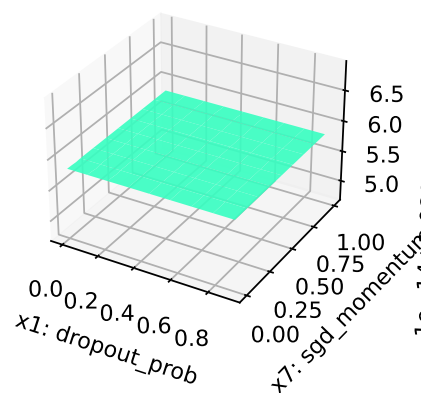
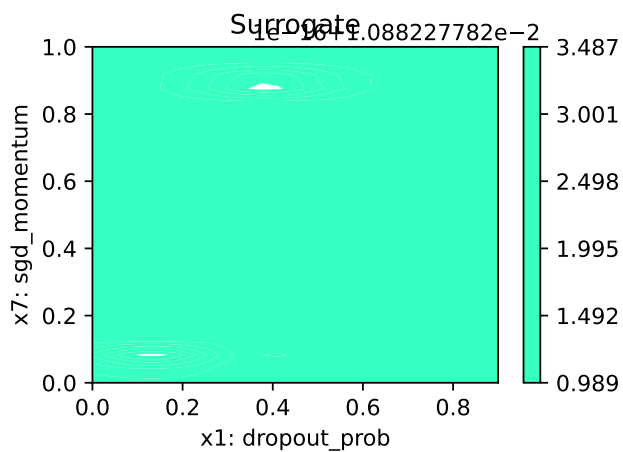
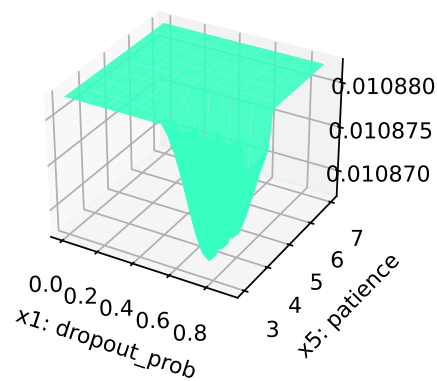
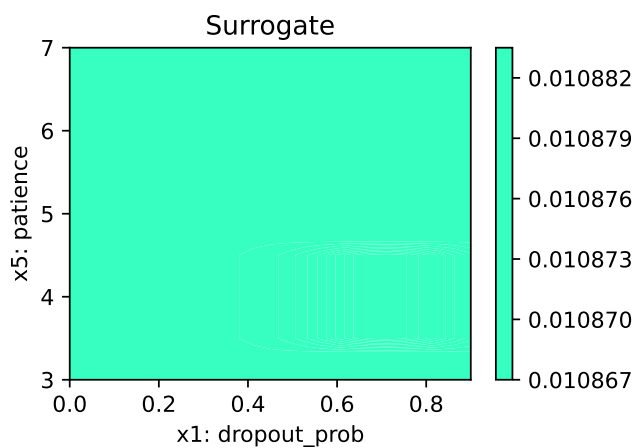
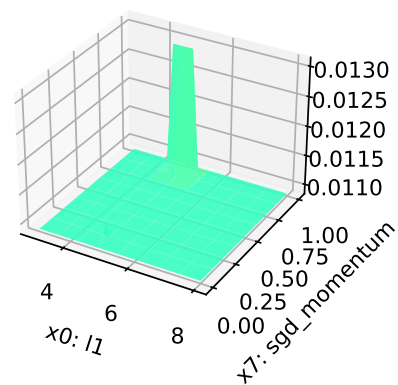
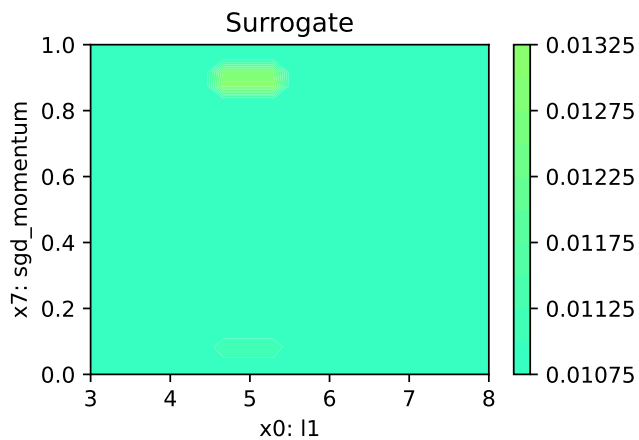
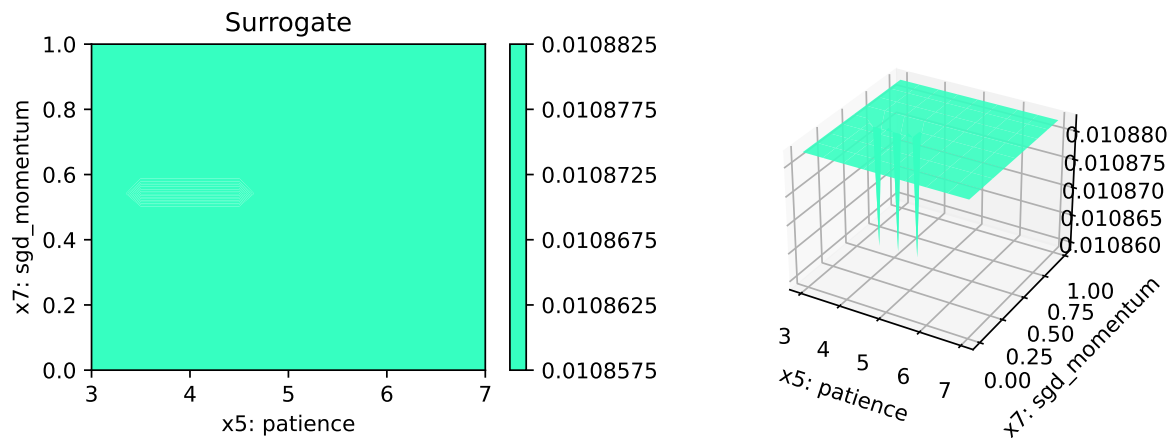


Figure 19.3: Contour plots.







19.10.5 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

Unable to display output for mime type(s): text/html

Parallel coordinates plots

Unable to display output for mime type(s): text/html

19.11 Summary and Outlook

This tutorial presents the hyperparameter tuning open source software **spotPython** for PyTorch. Some of the advantages of **spotPython** are:

- Numerical and categorical hyperparameters.
- Powerful surrogate models.
- Flexible approach and easy to use.
- Simple JSON files for the specification of the hyperparameters.
- Extension of default and user specified network classes.
- Noise handling techniques.
- Online visualization of the hyperparameter tuning process with **tensorboard**.

Currently, only rudimentary parallel and distributed neural network training is possible, but these capabilities will be extended in the future. The next version of **spotPython** will also include a more detailed documentation and more examples.

! Important

Important: This tutorial does not present a complete benchmarking study (Bartz-Beielstein et al. 2020). The results are only preliminary and highly dependent on the local configuration (hard- and software). Our goal is to provide a first impression of the performance of the hyperparameter tuning package **spotPython**. The results should be interpreted with care.

20 HPT: PyTorch With VBDP

In this tutorial, we will show how `spotPython` can be integrated into the PyTorch training workflow for a classification task.



Caution: Data must be downloaded manually

- Ensure that the corresponding data is available as `./data/VBDP/train.csv`.

This document refers to the following software versions:

- python: 3.10.10
- torch: 2.0.1
- torchvision: 0.15.0

```
pip list | grep "spot[RiverPython]"
```

<code>spotPython</code>	<code>0.2.52</code>
<code>spotRiver</code>	<code>0.0.94</code>

Note: you may need to restart the kernel to use updated packages.

`spotPython` can be installed via `pip`. Alternatively, the source code can be downloaded from `gitHub`: <https://github.com/sequential-parameter-optimization/spotPython>.

```
!pip install spotPython
```

- Uncomment the following lines if you want to for (re-)installation the latest version of `spotPython` from `gitHub`.

```
# import sys
# !{sys.executable} -m pip install --upgrade build
# !{sys.executable} -m pip install --upgrade --force-reinstall spotPython
```

20.1 Step 1: Setup

Before we consider the detailed experimental setup, we select the parameters that affect run time, initial design size and the device that is used.

 **Caution:** Run time and initial design size should be increased for real experiments

- MAX_TIME is set to one minute for demonstration purposes. For real experiments, this should be increased to at least 1 hour.
- INIT_SIZE is set to 5 for demonstration purposes. For real experiments, this should be increased to at least 10.

 **Note:** Device selection

- The device can be selected by setting the variable DEVICE.
- Since we are using a simple neural net, the setting "cpu" is preferred (on Mac).
- If you have a GPU, you can use "cuda:0" instead.
- If DEVICE is set to None, spotPython will automatically select the device.
 - This might result in "mps" on Macs, which is not the best choice for simple neural nets.

```
MAX_TIME = 1
INIT_SIZE = 5
DEVICE = None # "cpu" # "cuda:0"
```

```
from spotPython.utils.device import getDevice
DEVICE = getDevice(DEVICE)
print(DEVICE)
```

mps

```
import os
import copy
import socket
from datetime import datetime
from dateutil.tz import tzlocal
start_time = datetime.now(tzlocal())
HOSTNAME = socket.gethostname().split(".")[0]
experiment_name = '25-torch' + "_" + HOSTNAME + "_" + str(MAX_TIME) + "min_" + str(INIT_SIZE)
experiment_name = experiment_name.replace(':', '-')
```

```
print(experiment_name)
if not os.path.exists('./figures'):
    os.makedirs('./figures')
```

25-torch_maans03_1min_5init_2023-07-03_13-35-04

20.2 Step 2: Initialization of the fun_control Dictionary

 Caution: Tensorboard does not work under Windows

- Since tensorboard does not work under Windows, we recommend setting the parameter `tensorboard_path` to `None` if you are working under Windows.

`spotPython` uses a Python dictionary for storing the information required for the hyperparameter tuning process, which was described in Section 14.2, see [Initialization of the fun_control Dictionary](#) in the documentation.

```
from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="classification",
    tensorboard_path="runs/25_spot_torch_vbdp",
    device=DEVICE)
```

20.3 Step 3: PyTorch Data Loading

20.3.1 1. Load VBDP Data

```
import pandas as pd
from sklearn.preprocessing import OrdinalEncoder
train_df = pd.read_csv('./data/VBDP/train.csv')
# remove the id column
train_df = train_df.drop(columns=['id'])
n_samples = train_df.shape[0]
n_features = train_df.shape[1] - 1
target_column = "prognosis"
# Encode our prognosis labels as integers for easier decoding later
enc = OrdinalEncoder()
train_df[target_column] = enc.fit_transform(train_df[[target_column]])
```

```
# convert all entries to int for faster processing
train_df = train_df.astype(int)
```

- Add logical combinations (AND, OR, XOR) of the features to the data set:

```
from spotPython.utils.convert import add_logical_columns
df_new = train_df.copy()
# save the target column using "target_column" as the column name
target = train_df[target_column]
# remove the target column
df_new = df_new.drop(columns=[target_column])
train_df = add_logical_columns(df_new)
# add the target column back
train_df[target_column] = target
train_df = train_df.astype(int)
```

```
from sklearn.model_selection import train_test_split
import numpy as np
```

```
n_samples = train_df.shape[0]
n_features = train_df.shape[1] - 1
train_df.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
```

20.3.2 Check content of the target column

```
train_df[target_column].head()
```

prognosis	
0	3
1	7
2	3
3	10
4	6

```
X_train, X_test, y_train, y_test = train_test_split(train_df.drop(target_column, axis=1),
                                                    random_state=42,
                                                    test_size=0.25,
                                                    stratify=train_df[target_column])
```

```

trainset = pd.DataFrame(np.hstack((X_train, np.array(y_train).reshape(-1, 1))))
testset = pd.DataFrame(np.hstack((X_test, np.array(y_test).reshape(-1, 1))))
trainset.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
testset.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
print(train_df.shape)
print(trainset.shape)
print(testset.shape)

```

```
(707, 6113)
```

```
(530, 6113)
```

```
(177, 6113)
```

```

import torch
from sklearn.model_selection import train_test_split
from spotPython.torch.dataframedataset import DataFrameDataset
dtype_x = torch.float32
dtype_y = torch.long
train_df = DataFrameDataset(train_df, target_column=target_column, dtype_x=dtype_x, dtype_y=dtype_y)
train = DataFrameDataset(trainset, target_column=target_column, dtype_x=dtype_x, dtype_y=dtype_y)
test = DataFrameDataset(testset, target_column=target_column, dtype_x=dtype_x, dtype_y=dtype_y)
n_samples = len(train)

```

```

# add the dataset to the fun_control
fun_control.update({"data": train_df, # full dataset,
                  "train": train,
                  "test": test,
                  "n_samples": n_samples,
                  "target_column": target_column})

```

20.4 Step 4: Specification of the Preprocessing Model

After the training and test data are specified and added to the `fun_control` dictionary, `spotPython` allows the specification of a data preprocessing pipeline, e.g., for the scaling of the data or for the one-hot encoding of categorical variables, see Section 14.4. This feature is not used here, so we do not change the default value (which is `None`).

20.5 Step 5: Select algorithm and core_model_hyper_dict

20.5.1 Implementing a Configurable Neural Network With spotPython

spotPython includes the `Net_vbdp` class which is implemented in the file `netvbdp.py`. The class is imported here.

This class inherits from the class `Net_Core` which is implemented in the file `netcore.py`, see Section [14.5.1](#).

20.5.2 Add the NN Model to the fun_control Dictionary

```
from spotPython.torch.netvbdp import Net_vbdp
from spotPython.data.torch_hyper_dict import TorchHyperDict
from spotPython.hyperparameters.values import add_core_model_to_fun_control
fun_control = add_core_model_to_fun_control(core_model=Net_vbdp,
                                           fun_control=fun_control,
                                           hyper_dict=TorchHyperDict)
```

The corresponding entries for the `core_model` class are shown below.

```
fun_control['core_model_hyper_dict']
```

```
{'_L0': {'type': 'int',
        'default': 64,
        'transform': 'None',
        'lower': 64,
        'upper': 64},
 'l1': {'type': 'int',
        'default': 8,
        'transform': 'transform_power_2_int',
        'lower': 8,
        'upper': 16},
 'dropout_prob': {'type': 'float',
                  'default': 0.01,
                  'transform': 'None',
                  'lower': 0.0,
                  'upper': 0.9},
 'lr_mult': {'type': 'float',
             'default': 1.0,
             'transform': 'None',
```



```

'lower': 0.1,
'upper': 10.0},
'batch_size': {'type': 'int',
'default': 4,
'transform': 'transform_power_2_int',
'lower': 1,
'upper': 4},
'epochs': {'type': 'int',
'default': 4,
'transform': 'transform_power_2_int',
'lower': 4,
'upper': 9},
'k_folds': {'type': 'int',
'default': 1,
'transform': 'None',
'lower': 1,
'upper': 1},
'patience': {'type': 'int',
'default': 2,
'transform': 'transform_power_2_int',
'lower': 1,
'upper': 5},
'optimizer': {'levels': ['Adadelata',
'Adagrad',
'Adam',
'AdamW',
'SparseAdam',
'Adamax',
'ASGD',
'NAdam',
'RAdam',
'RMSprop',
'Rprop',
'SGD'],
'type': 'factor',
'default': 'SGD',
'transform': 'None',
'class_name': 'torch.optim',
'core_model_parameter_type': 'str',
'lower': 0,
'upper': 12},
'sgd_momentum': {'type': 'float',
'default': 0.0,

```

```
'transform': 'None',
'lower': 0.0,
'upper': 1.0}}
```

20.6 Step 6: Modify hyper_dict Hyperparameters for the Selected Algorithm aka core_model

spotPython provides functions for modifying the hyperparameters, their bounds and factors as well as for activating and de-activating hyperparameters without re-compilation of the Python source code. These functions were described in Section 14.6.

 Caution: Small number of epochs for demonstration purposes

- `epochs` and `patience` are set to small values for demonstration purposes. These values are too small for a real application.
- More resonable values are, e.g.:
 - `fun_control = modify_hyper_parameter_bounds(fun_control, "epochs", bounds=[7, 9])` and
 - `fun_control = modify_hyper_parameter_bounds(fun_control, "patience", bounds=[2, 7])`

```
from spotPython.hyperparameters.values import modify_hyper_parameter_bounds
```

```
fun_control = modify_hyper_parameter_bounds(fun_control, "_L0", bounds=[n_features, n_feat
fun_control = modify_hyper_parameter_bounds(fun_control, "l1", bounds=[6, 13])
fun_control = modify_hyper_parameter_bounds(fun_control, "epochs", bounds=[2, 3])
fun_control = modify_hyper_parameter_bounds(fun_control, "patience", bounds=[2, 2])
```

```
from spotPython.hyperparameters.values import modify_hyper_parameter_levels
```

```
fun_control = modify_hyper_parameter_levels(fun_control, "optimizer", ["Adam", "AdamW", "Ad
# fun_control = modify_hyper_parameter_levels(fun_control, "optimizer", ["Adam"])
# fun_control["core_model_hyper_dict"]
```

20.6.1 Optimizers

Optimizers are described in Section 14.6.1.

```

fun_control = modify_hyper_parameter_bounds(fun_control,
    "lr_mult", bounds=[1e-3, 1e-3])
fun_control = modify_hyper_parameter_bounds(fun_control,
    "sgd_momentum", bounds=[0.9, 0.9])

```

20.7 Step 7: Selection of the Objective (Loss) Function

20.7.1 Evaluation

The evaluation procedure requires the specification of two elements:

1. the way how the data is split into a train and a test set (see Section [14.7.1](#))
2. the loss function (and a metric).

20.7.2 Loss Functions and Metrics

The loss function is specified by the key "loss_function". We will use CrossEntropy loss for the multiclass-classification task.

```

from torch.nn import CrossEntropyLoss
loss_function = CrossEntropyLoss()
fun_control.update({"loss_function": loss_function})

```

20.7.3 Metric

- We will use the MAP@k metric for the evaluation of the model. Here is an example how this metric is calculated.

```

from spotPython.torch.mapk import MAPK
import torch
mapk = MAPK(k=2)
target = torch.tensor([0, 1, 2, 2])
preds = torch.tensor(
    [
        [0.5, 0.2, 0.2], # 0 is in top 2
        [0.3, 0.4, 0.2], # 1 is in top 2
        [0.2, 0.4, 0.3], # 2 is in top 2
        [0.7, 0.2, 0.1], # 2 isn't in top 2
    ]
)

```

```

    ]
)
mapk.update(preds, target)
print(mapk.compute()) # tensor(0.6250)

```

tensor(0.6250)

```

from spotPython.torch.mapk import MAPK
import torchmetrics
metric_torch = MAPK(k=3)
fun_control.update({"metric_torch": metric_torch})

```

20.8 Step 8: Calling the SPOT Function

20.8.1 Preparing the SPOT Call

The following code passes the information about the parameter ranges and bounds to `spot`.

```

# extract the variable types, names, and bounds
from spotPython.hyperparameters.values import (get_bound_values,
    get_var_name,
    get_var_type,)
var_type = get_var_type(fun_control)
var_name = get_var_name(fun_control)
fun_control.update({"var_type": var_type,
    "var_name": var_name})
lower = get_bound_values(fun_control, "lower")
upper = get_bound_values(fun_control, "upper")

```

Now, the dictionary `fun_control` contains all information needed for the hyperparameter tuning. Before the hyperparameter tuning is started, it is recommended to take a look at the experimental design. The method `gen_design_table` generates a design table as follows:

```

from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control))

```

name	type	default	lower	upper	transform
-----	-----	-----	-----	-----	-----

_L0	int	64	6112	6112	None	
l1	int	8	6	13	transform_power_2_int	
dropout_prob	float	0.01	0	0.9	None	
lr_mult	float	1.0	0.001	0.001	None	
batch_size	int	4	1	4	transform_power_2_int	
epochs	int	4	2	3	transform_power_2_int	
k_folds	int	1	1	1	None	
patience	int	2	2	2	transform_power_2_int	
optimizer	factor	SGD	0	3	None	
sgd_momentum	float	0.0	0.9	0.9	None	

This allows to check if all information is available and if the information is correct.

20.8.2 The Objective Function `fun_torch`

The objective function `fun_torch` is selected next. It implements an interface from PyTorch's training, validation, and testing methods to `spotPython`.

```
from spotPython.fun.hypertorch import HyperTorch
fun = HyperTorch().fun_torch

from spotPython.hyperparameters.values import get_default_hyperparameters_as_array
hyper_dict=TorchHyperDict().load()
X_start = get_default_hyperparameters_as_array(fun_control, hyper_dict)
```

20.8.3 Starting the Hyperparameter Tuning

The `spotPython` hyperparameter tuning is started by calling the `Spot` function as described in Section 14.8.4.

```
import numpy as np
from spotPython.spot import spot
from math import inf
spot_tuner = spot.Spot(fun=fun,
                      lower = lower,
                      upper = upper,
                      fun_evals = inf,
                      fun_repeats = 1,
                      max_time = MAX_TIME,
                      noise = False,
```

```

tolerance_x = np.sqrt(np.spacing(1)),
var_type = var_type,
var_name = var_name,
infill_criterion = "y",
n_points = 1,
seed=123,
log_level = 50,
show_models= False,
show_progress= True,
fun_control = fun_control,
design_control={"init_size": INIT_SIZE,
               "repeats": 1},
surrogate_control={"noise": True,
                  "cod_type": "norm",
                  "min_theta": -4,
                  "max_theta": 3,
                  "n_theta": len(var_name),
                  "model_fun_evals": 10_000,
                  "log_level": 50
                })

spot_tuner.run(X_start=X_start)

```

config: {'_L0': 6112, 'l1': 2048, 'dropout_prob': 0.17031221661559992, 'lr_mult': 0.001, 'ba

Epoch: 1 |

MAPK: 0.1793154776096344 | Loss: 2.3979182073048184 | Acc: 0.1273584905660377.

Epoch: 2 |

MAPK: 0.1845238208770752 | Loss: 2.3979019778115407 | Acc: 0.1320754716981132.

Epoch: 3 |

MAPK: 0.2008928507566452 | Loss: 2.3978130817413330 | Acc: 0.1367924528301887.

Epoch: 4 |

MAPK: 0.1964285671710968 | Loss: 2.3978581598826816 | Acc: 0.1415094339622641.

Epoch: 5 |

MAPK: 0.2142857164144516 | Loss: 2.3978352035794939 | Acc: 0.1509433962264151.

Epoch: 6 |

MAPK: 0.2105654776096344 | Loss: 2.3977752923965454 | Acc: 0.1320754716981132.
Epoch: 7 |

MAPK: 0.2343750000000000 | Loss: 2.3977158580507552 | Acc: 0.1698113207547170.
Epoch: 8 |

MAPK: 0.2232142686843872 | Loss: 2.3976944174085344 | Acc: 0.1415094339622641.
Returned to Spot: Validation loss: 2.3976944174085344

config: {'_L0': 6112, 'l1': 256, 'dropout_prob': 0.19379790035512987, 'lr_mult': 0.001, 'bat
Epoch: 1 |

MAPK: 0.1712962985038757 | Loss: 2.3980199672557689 | Acc: 0.0990566037735849.
Epoch: 2 |

MAPK: 0.1689814776182175 | Loss: 2.3979500134785972 | Acc: 0.0943396226415094.
Epoch: 3 |

MAPK: 0.1666666716337204 | Loss: 2.3979757538548223 | Acc: 0.0849056603773585.
Epoch: 4 |

MAPK: 0.1550925970077515 | Loss: 2.3980355969181768 | Acc: 0.0613207547169811.
Returned to Spot: Validation loss: 2.398035596918177

config: {'_L0': 6112, 'l1': 4096, 'dropout_prob': 0.6759063718076167, 'lr_mult': 0.001, 'bat
Epoch: 1 |

MAPK: 0.1533019095659256 | Loss: 2.3977285038750127 | Acc: 0.0707547169811321.
Epoch: 2 |

MAPK: 0.1627358347177505 | Loss: 2.3976753702703513 | Acc: 0.0896226415094340.
Epoch: 3 |

MAPK: 0.1910377293825150 | Loss: 2.3973205426953874 | Acc: 0.1179245283018868.
Epoch: 4 |

MAPK: 0.1816037744283676 | Loss: 2.3971340116464868 | Acc: 0.0990566037735849.
Epoch: 5 |

MAPK: 0.2114779651165009 | Loss: 2.3967223572281173 | Acc: 0.1179245283018868.
Epoch: 6 |

MAPK: 0.1957547366619110 | Loss: 2.3965382081157758 | Acc: 0.0990566037735849.
Epoch: 7 |

MAPK: 0.2256288826465607 | Loss: 2.3956945887151755 | Acc: 0.1273584905660377.
Epoch: 8 |

MAPK: 0.2130502909421921 | Loss: 2.3951818313238755 | Acc: 0.1226415094339623.
Returned to Spot: Validation loss: 2.3951818313238755

config: {'_L0': 6112, 'l1': 128, 'dropout_prob': 0.37306669346546995, 'lr_mult': 0.001, 'batch_size': 128}
Epoch: 1 |

MAPK: 0.1470125913619995 | Loss: 2.3993017583523155 | Acc: 0.0896226415094340.
Epoch: 2 |

MAPK: 0.1470125913619995 | Loss: 2.3992382625363908 | Acc: 0.0896226415094340.
Epoch: 3 |

MAPK: 0.1470125913619995 | Loss: 2.3992580557769201 | Acc: 0.0896226415094340.
Epoch: 4 |

MAPK: 0.1446540951728821 | Loss: 2.3991979338088125 | Acc: 0.0849056603773585.
Returned to Spot: Validation loss: 2.3991979338088125

config: {'_L0': 6112, 'l1': 1024, 'dropout_prob': 0.870137281216666, 'lr_mult': 0.001, 'batch_size': 128}
Epoch: 1 |

MAPK: 0.1288580447435379 | Loss: 2.3975733032932989 | Acc: 0.0424528301886792.
Epoch: 2 |

MAPK: 0.1682098656892776 | Loss: 2.3977194273913347 | Acc: 0.0943396226415094.
Epoch: 3 |

MAPK: 0.1743827015161514 | Loss: 2.3975409225181297 | Acc: 0.0707547169811321.
Epoch: 4 |

MAPK: 0.1712962985038757 | Loss: 2.3976151325084545 | Acc: 0.0990566037735849.
Epoch: 5 |

MAPK: 0.1797839254140854 | Loss: 2.3974891680258290 | Acc: 0.0896226415094340.
Epoch: 6 |

MAPK: 0.1628086417913437 | Loss: 2.3976068055188215 | Acc: 0.0849056603773585.
Epoch: 7 |

MAPK: 0.1658950597047806 | Loss: 2.3975864074848316 | Acc: 0.0896226415094340.
Epoch: 8 |

MAPK: 0.1604938358068466 | Loss: 2.3976810243394642 | Acc: 0.0801886792452830.
Returned to Spot: Validation loss: 2.397681024339464

config: {'_L0': 6112, 'l1': 4096, 'dropout_prob': 0.4132005099912892, 'lr_mult': 0.001, 'bat

Epoch: 1 |

MAPK: 0.2044024914503098 | Loss: 2.3973671445306741 | Acc: 0.1226415094339623.
Epoch: 2 |

MAPK: 0.2295597344636917 | Loss: 2.3970097910683110 | Acc: 0.1226415094339623.
Epoch: 3 |

MAPK: 0.2397798150777817 | Loss: 2.3966489625426957 | Acc: 0.1226415094339623.
Epoch: 4 |

MAPK: 0.2649370431900024 | Loss: 2.3960055427731208 | Acc: 0.1226415094339623.
Epoch: 5 |

MAPK: 0.2547169029712677 | Loss: 2.3953653956359289 | Acc: 0.1226415094339623.
Epoch: 6 |

MAPK: 0.2720125317573547 | Loss: 2.3940943771938108 | Acc: 0.1226415094339623.
Epoch: 7 |

MAPK: 0.2397798299789429 | Loss: 2.3928466540462567 | Acc: 0.1226415094339623.
Epoch: 8 |

```
MAPK: 0.2350628525018692 | Loss: 2.3911897168969207 | Acc: 0.1226415094339623.  
Returned to Spot: Validation loss: 2.3911897168969207  
spotPython tuning: 2.3911897168969207 [#####--] 82.02%
```

```
config: {'_L0': 6112, 'l1': 512, 'dropout_prob': 0.4054506390535282, 'lr_mult': 0.001, 'batch_size': 128,  
Epoch: 1 |
```

```
MAPK: 0.1533019095659256 | Loss: 2.3983700455359691 | Acc: 0.0801886792452830.  
Epoch: 2 |
```

```
MAPK: 0.1470126062631607 | Loss: 2.3983595596169525 | Acc: 0.0801886792452830.  
Epoch: 3 |
```

```
MAPK: 0.1462264358997345 | Loss: 2.3983060413936399 | Acc: 0.0801886792452830.  
Epoch: 4 |
```

```
MAPK: 0.1540880799293518 | Loss: 2.3982332292592750 | Acc: 0.0801886792452830.  
Epoch: 5 |
```

```
MAPK: 0.1517295837402344 | Loss: 2.3982075970127896 | Acc: 0.0801886792452830.  
Epoch: 6 |
```

```
MAPK: 0.1501572579145432 | Loss: 2.3981801293930918 | Acc: 0.0801886792452830.  
Epoch: 7 |
```

```
MAPK: 0.1580188870429993 | Loss: 2.3981470701829442 | Acc: 0.0801886792452830.  
Epoch: 8 |
```

```
MAPK: 0.1627358645200729 | Loss: 2.3980459537146226 | Acc: 0.0801886792452830.  
Returned to Spot: Validation loss: 2.3980459537146226  
spotPython tuning: 2.3911897168969207 [#####] 100.00% Done...
```

```
<spotPython.spot.spot.Spot at 0x18f54f3a0>
```

20.9 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section 14.9, see also the description in the documentation: [Tensorboard](#).

20.10 Step 10: Results

After the hyperparameter tuning run is finished, the results can be analyzed as described in Section 14.10.

```
spot_tuner.plot_progress(log_y=False,
                        filename="./figures/" + experiment_name+"_progress.png")
```

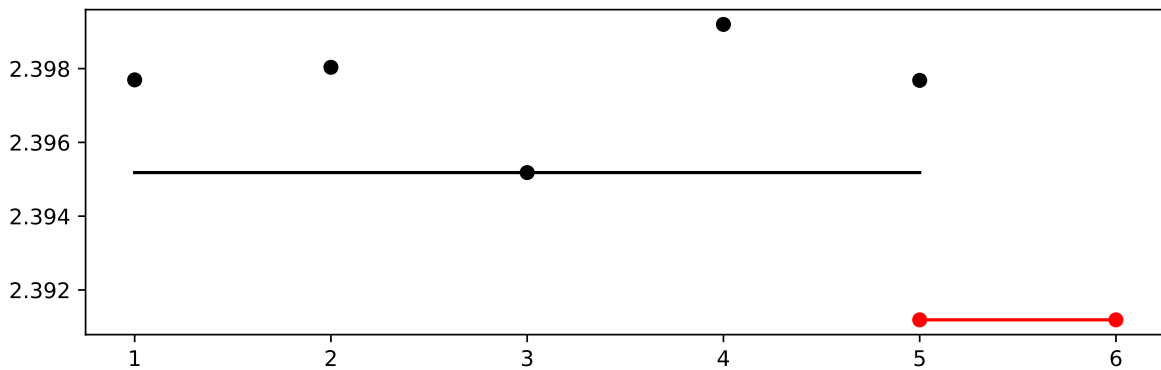


Figure 20.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

```
from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control=fun_control, spot=spot_tuner))
```

name	type	default	lower	upper	tuned	transform
_L0	int	64	6112.0	6112.0	6112.0	None
l1	int	8	6.0	13.0	12.0	transform_pow
dropout_prob	float	0.01	0.0	0.9	0.4132005099912892	None
lr_mult	float	1.0	0.001	0.001	0.001	None
batch_size	int	4	1.0	4.0	1.0	transform_pow
epochs	int	4	2.0	3.0	3.0	transform_pow
k_folds	int	1	1.0	1.0	1.0	None
patience	int	2	2.0	2.0	2.0	transform_pow
optimizer	factor	SGD	0.0	3.0	3.0	None
sgd_momentum	float	0.0	0.9	0.9	0.9	None

```
spot_tuner.plot_importance(threshold=0.025,
                           filename="./figures/" + experiment_name+"_importance.png")
```

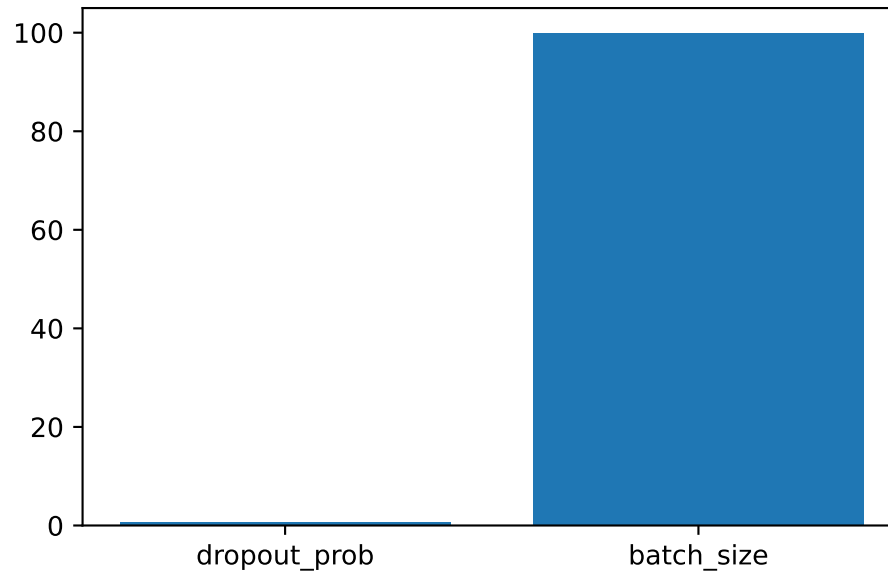


Figure 20.2: Variable importance plot, threshold 0.025.

20.10.1 Get the Tuned Architecture

```
from spotPython.hyperparameters.values import get_one_core_model_from_X
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
model_spot = get_one_core_model_from_X(X, fun_control)
model_spot
```

```
Net_vbdp(
  (fc1): Linear(in_features=6112, out_features=4096, bias=True)
  (fc2): Linear(in_features=4096, out_features=2048, bias=True)
  (fc3): Linear(in_features=2048, out_features=1024, bias=True)
  (fc4): Linear(in_features=1024, out_features=512, bias=True)
  (fc5): Linear(in_features=512, out_features=11, bias=True)
  (relu): ReLU()
  (softmax): Softmax(dim=1)
  (dropout1): Dropout(p=0.4132005099912892, inplace=False)
  (dropout2): Dropout(p=0.2066002549956446, inplace=False)
)
```

20.10.2 Evaluation of the Tuned Architecture

```
from spotPython.torch.traintest import (
    train_tuned,
    test_tuned,
)
train_tuned(net=model_spot, train_dataset=train,
            loss_function=fun_control["loss_function"],
            metric=fun_control["metric_torch"],
            shuffle=True,
            device = fun_control["device"],
            path=None,
            task=fun_control["task"],)
```

Epoch: 1 |

MAPK: 0.1776729971170425 | Loss: 2.3976604848537804 | Acc: 0.0707547169811321.

Epoch: 2 |

MAPK: 0.1658805310726166 | Loss: 2.3973652394312732 | Acc: 0.0660377358490566.

Epoch: 3 |

MAPK: 0.1721698343753815 | Loss: 2.3971810025988884 | Acc: 0.0849056603773585.

Epoch: 4 |

MAPK: 0.1933962255716324 | Loss: 2.3963526172458001 | Acc: 0.1132075471698113.

Epoch: 5 |

MAPK: 0.1878930777311325 | Loss: 2.3954812670653722 | Acc: 0.1132075471698113.

Epoch: 6 |

MAPK: 0.1902516037225723 | Loss: 2.3941759433386460 | Acc: 0.1084905660377359.

Epoch: 7 |

MAPK: 0.1878930926322937 | Loss: 2.3928982806655594 | Acc: 0.1084905660377359.

Epoch: 8 |

MAPK: 0.2012578696012497 | Loss: 2.3909529492540180 | Acc: 0.1179245283018868.

Returned to Spot: Validation loss: 2.390952949254018

If `path` is set to a filename, e.g., `path = "model_spot_trained.pt"`, the weights of the trained model will be loaded from this file.

```
test_tuned(net=model_spot, test_dataset=test,
           shuffle=False,
           loss_function=fun_control["loss_function"],
           metric=fun_control["metric_torch"],
           device = fun_control["device"],
           task=fun_control["task"],)
```

MAPK: 0.2331460714340210 | Loss: 2.3826623846975603 | Acc: 0.1299435028248588.

Final evaluation: Validation loss: 2.3826623846975603

Final evaluation: Validation metric: 0.233146071434021

(2.3826623846975603, nan, tensor(0.2331))

20.10.3 Cross-validated Evaluations

- This is the evaluation that will be used in the comparison.

Caution: Cross-validated Evaluations

- The number of folds is set to 1 by default.
- Here it was changed to 3 for demonstration purposes.
- Set the number of folds to a reasonable value, e.g., 10.
- This can be done by setting the `k_folds` attribute of the model as follows:
- `setattr(model_spot, "k_folds", 10)`

```
from spotPython.torch.traintest import evaluate_cv
# modify k-folds:
setattr(model_spot, "k_folds", 3)
df_eval, df_preds, df_metrics = evaluate_cv(net=model_spot,
      dataset=fun_control["data"],
      loss_function=fun_control["loss_function"],
      metric=fun_control["metric_torch"],
      task=fun_control["task"],
      writer=fun_control["writer"],
      writerId="model_spot_cv",
      device = fun_control["device"])
```

Fold: 1

Epoch: 1 |

MAPK: 0.1645480394363403 | Loss: 2.3976494235507513 | Acc: 0.1059322033898305.

Epoch: 2 |

MAPK: 0.1793785095214844 | Loss: 2.3972296149043713 | Acc: 0.1186440677966102.

Epoch: 3 |

MAPK: 0.2394067347049713 | Loss: 2.3964333251371222 | Acc: 0.1525423728813559.

Epoch: 4 |

MAPK: 0.2648304998874664 | Loss: 2.3949893369513044 | Acc: 0.1694915254237288.

Epoch: 5 |

MAPK: 0.2881355881690979 | Loss: 2.3928930355330644 | Acc: 0.1906779661016949.

Epoch: 6 |

MAPK: 0.3156779110431671 | Loss: 2.3891845072730113 | Acc: 0.2288135593220339.

Epoch: 7 |

MAPK: 0.3340395390987396 | Loss: 2.3839198573160978 | Acc: 0.2415254237288136.

Epoch: 8 |

MAPK: 0.3495762944221497 | Loss: 2.3775227857848344 | Acc: 0.2627118644067797.

Fold: 2

Epoch: 1 |

MAPK: 0.1913841664791107 | Loss: 2.3970539893134166 | Acc: 0.1186440677966102.

Epoch: 2 |

MAPK: 0.2231637984514236 | Loss: 2.3965012562476984 | Acc: 0.1186440677966102.

Epoch: 3 |

MAPK: 0.2499999552965164 | Loss: 2.3951188224857138 | Acc: 0.1186440677966102.

Epoch: 4 |

MAPK: 0.2344632297754288 | Loss: 2.3933223768816156 | Acc: 0.1186440677966102.

Epoch: 5 |

MAPK: 0.2182203084230423 | Loss: 2.3901682970887523 | Acc: 0.1186440677966102.
Epoch: 6 |

MAPK: 0.2083333283662796 | Loss: 2.3876159878100380 | Acc: 0.1186440677966102.
Epoch: 7 |

MAPK: 0.2252824753522873 | Loss: 2.3847315937785778 | Acc: 0.1186440677966102.
Epoch: 8 |

MAPK: 0.2492937594652176 | Loss: 2.3822026192131691 | Acc: 0.1186440677966102.
Fold: 3
Epoch: 1 |

MAPK: 0.1716101765632629 | Loss: 2.3976111533278126 | Acc: 0.0851063829787234.
Epoch: 2 |

MAPK: 0.1977400630712509 | Loss: 2.3969947241120417 | Acc: 0.1148936170212766.
Epoch: 3 |

MAPK: 0.2055084407329559 | Loss: 2.3959967362678656 | Acc: 0.1361702127659574.
Epoch: 4 |

MAPK: 0.1991525292396545 | Loss: 2.3944945173748469 | Acc: 0.1361702127659574.
Epoch: 5 |

MAPK: 0.1963276714086533 | Loss: 2.3925338862305980 | Acc: 0.1276595744680851.
Epoch: 6 |

MAPK: 0.1899717301130295 | Loss: 2.3886827089018743 | Acc: 0.1191489361702128.
Epoch: 7 |

MAPK: 0.1998587250709534 | Loss: 2.3855099698244513 | Acc: 0.1361702127659574.
Epoch: 8 |

MAPK: 0.2019773721694946 | Loss: 2.3827925758846735 | Acc: 0.1404255319148936.

```
metric_name = type(fun_control["metric_torch"]).__name__  
print(f"loss: {df_eval}, Cross-validated {metric_name}: {df_metrics}")
```

loss: 2.3808393269608925, Cross-validated MAPK: 0.2669491469860077

20.10.4 Detailed Hyperparameter Plots

```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

dropout_prob: 0.7261315500302485
batch_size: 100.0

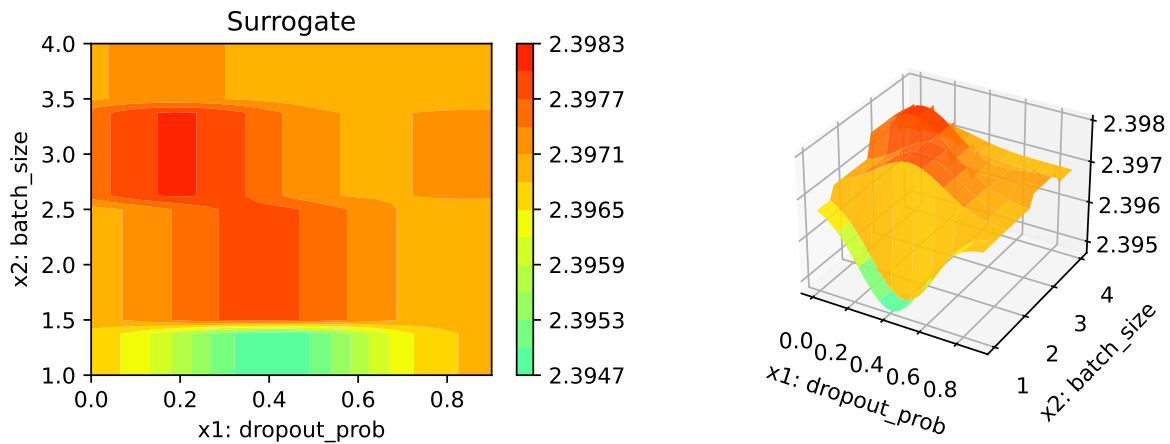


Figure 20.3: Contour plots.

20.10.5 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

Unable to display output for mime type(s): text/html

Parallel coordinates plots

Unable to display output for mime type(s): text/html

```
# close tensorboard writer
if fun_control["writer"] is not None:
    fun_control["writer"].close()
```

20.10.6 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

```
PLOT_ALL = False
if PLOT_ALL:
    n = spot_tuner.k
    for i in range(n-1):
        for j in range(i+1, n):
            spot_tuner.plot_contour(i=i, j=j, min_z=min_z, max_z = max_z)
```

21 HPT PyTorch Lightning: VBDP

In this tutorial, we will show how `spotPython` can be integrated into the PyTorch Lightning training workflow for a classification task.



Caution: Data must be downloaded manually

- Ensure that the corresponding data is available as `./data/VBDP/train.csv`.

This document refers to the following software versions:

- python: 3.10.10
- torch: 2.0.1
- torchvision: 0.15.0

```
pip list | grep "spot[RiverPython]"
```

spotPython	0.2.52
spotRiver	0.0.94

Note: you may need to restart the kernel to use updated packages.

`spotPython` can be installed via `pip`. Alternatively, the source code can be downloaded from `gitHub`: <https://github.com/sequential-parameter-optimization/spotPython>.

```
!pip install spotPython
```

- Uncomment the following lines if you want to for (re-)installation the latest version of `spotPython` from `GitHub`.

```
# import sys
# !{sys.executable} -m pip install --upgrade build
# !{sys.executable} -m pip install --upgrade --force-reinstall spotPython
```

21.1 Step 1: Setup

Before we consider the detailed experimental setup, we select the parameters that affect run time, initial design size and the device that is used.

 **Caution:** Run time and initial design size should be increased for real experiments

- `MAX_TIME` is set to one minute for demonstration purposes. For real experiments, this should be increased to at least 1 hour.
- `INIT_SIZE` is set to 5 for demonstration purposes. For real experiments, this should be increased to at least 10.
- `WORKERS` is set to 0 for demonstration purposes. For real experiments, this should be increased. See the warnings that are printed when the number of workers is set to 0.

 **Note:** Device selection

- The device can be selected by setting the variable `DEVICE`.
- Since we are using a simple neural net, the setting `"cpu"` is preferred (on Mac).
- If you have a GPU, you can use `"cuda:0"` instead.
- If `DEVICE` is set to `"auto"` or `None`, `spotPython` will automatically select the device.
 - This might result in `"mps"` on Macs, which is not the best choice for simple neural nets.

 **Note:** Prefix

- The prefix `PREFIX` is used for the experiment name and the name of the log file.

```
MAX_TIME = 1
INIT_SIZE = 5
DEVICE = "cpu" # "cpu" # "cuda:0"
WORKERS = 0
PREFIX="31"
```

```
from spotPython.utils.device import getDevice
DEVICE = getDevice(DEVICE)
print(DEVICE)
```

cpu

```
import os
if not os.path.exists('./figures'):
    os.makedirs('./figures')
```

21.2 Step 2: Initialization of the `fun_control` Dictionary

 Caution: Tensorboard does not work under Windows

- Since tensorboard does not work under Windows, we recommend setting the parameter `tensorboard_path` to `None` if you are working under Windows.

`spotPython` uses a Python dictionary for storing the information required for the hyperparameter tuning process, which was described in Section 14.2, see [Initialization of the `fun_control` Dictionary](#) in the documentation.

```
from spotPython.utils.init import fun_control_init
from spotPython.utils.file import get_experiment_name
experiment_name = get_experiment_name(prefix=PREFIX)
fun_control = fun_control_init(
    num_workers=WORKERS,
    device=DEVICE,
    _L_in=64,
    _L_out=11)
```

21.3 Step 3: PyTorch Data Loading

21.3.1 Lightning Dataset and DataModule

The data loading and preprocessing is handled by `Lightning` and `PyTorch`. It comprehends the following classes:

- `CSVDataset`: A class that loads the data from a CSV file. [\[SOURCE\]](#)
- `CSVDataModule`: A class that prepares the data for training and testing. [\[SOURCE\]](#)

21.3.1.1 Taking a Look at the Data

```
import torch
from spotPython.light.csvdataset import CSVDataset
from torch.utils.data import DataLoader
from torchvision.transforms import ToTensor

# Create an instance of CSVDataset
dataset = CSVDataset(csv_file="./data/VBDP/train.csv", train=True)
# show the dimensions of the input data
print(dataset[0][0].shape)
# show the first element of the input data
print(dataset[0][0])
# show the size of the dataset
print(f"Dataset Size: {len(dataset)}")
```

```
torch.Size([64])
tensor([1., 1., 0., 1., 1., 1., 1., 0., 1., 1., 1., 1., 0., 0., 1., 1., 0., 0.,
        1., 0., 1., 0., 1., 1., 1., 1., 1., 1., 0., 0., 1., 0., 0., 0., 0.,
        1., 0., 0., 0., 0., 0., 1., 0., 1., 0., 1., 0., 0., 0., 0., 1., 0., 1.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
Dataset Size: 707
```

```
# Set batch size for DataLoader
batch_size = 3
# Create DataLoader
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)

# Iterate over the data in the DataLoader
for batch in dataloader:
    inputs, targets = batch
    print(f"Batch Size: {inputs.size(0)}")
    print("-----")
    print(f"Inputs: {inputs}")
    print(f"Targets: {targets}")
    break
```

Batch Size: 3

Inputs: tensor([[1., 1., 1., 1., 0., 1., 0., 1., 0., 1., 1., 0., 1., 0., 1., 0., 0., 0.,

```

0., 1., 1., 1., 1., 0., 1., 1., 1., 1., 0., 1., 0., 0., 0., 1., 1., 1.,
1., 1., 0., 1., 0., 1., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
[0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1.,
0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 1., 0., 1., 1., 1., 1.,
1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
[1., 0., 0., 1., 1., 0., 1., 1., 0., 1., 1., 1., 1., 0., 0., 1., 0., 0.,
1., 0., 1., 1., 1., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]]

```

Targets: tensor([1, 4, 1])

Caution: Data Loading in Lightning

- Data loading is handled independently from the `fun_control` dictionary by `Lightning` and `PyTorch`.
- In contrast to `spotPython` with `torch`, `river` and `sklearn`, the data sets are not added to the `fun_control` dictionary.

21.4 Step 4: Specification of the Preprocessing Model

The `fun_control` dictionary, the `torch`, `sklearn` and `river` versions of `spotPython` allow the specification of a data preprocessing pipeline, e.g., for the scaling of the data or for the one-hot encoding of categorical variables, see Section 14.4. This feature is not used in the `Lightning` version.

Caution: Data preprocessing in Lightning

`Lightning` allows the data preprocessing to be specified in the `LightningDataModule` class. It is not considered here, because it should be computed at one location only.

21.5 Step 5: Select the NN Model (algorithm) and `core_model_hyper_dict`

21.5.1 Implementing a Configurable Neural Network With `spotPython`

`spotPython` includes the `NetLightBase` class [\[SOURCE\]](#) for configurable neural networks. The class is imported here. It inherits from the class `Lightning.LightningModule`, which

is the base class for all models in Lightning. `Lightning.LightningModule` is a subclass of `torch.nn.Module` and provides additional functionality for the training and testing of neural networks. The class `Lightning.LightningModule` is described in the [Lightning documentation](#).

21.5.2 Add the NN Model to the `fun_control` Dictionary

```
from spotPython.light.netlightbase import NetLightBase
from spotPython.data.light_hyper_dict import LightHyperDict
from spotPython.hyperparameters.values import add_core_model_to_fun_control
fun_control = add_core_model_to_fun_control(core_model=NetLightBase,
                                          fun_control=fun_control,
                                          hyper_dict= LightHyperDict)
```

The default entries for the `core_model` class are shown below.

```
fun_control['core_model_hyper_dict']
```

```
{'l1': {'type': 'int',
        'default': 3,
        'transform': 'transform_power_2_int',
        'lower': 3,
        'upper': 8},
 'epochs': {'type': 'int',
             'default': 4,
             'transform': 'transform_power_2_int',
             'lower': 4,
             'upper': 9},
 'batch_size': {'type': 'int',
                 'default': 4,
                 'transform': 'transform_power_2_int',
                 'lower': 1,
                 'upper': 4},
 'act_fn': {'levels': ['Sigmoid', 'Tanh', 'ReLU', 'LeakyReLU', 'ELU', 'Swish'],
            'type': 'factor',
            'default': 'ReLU',
            'transform': 'None',
            'class_name': 'spotPython.torch.activation',
            'core_model_parameter_type': 'instance()',
            'lower': 0,
            'upper': 5},
```



```

'optimizer': {'levels': ['Adadelata',
    'Adagrad',
    'Adam',
    'AdamW',
    'SparseAdam',
    'Adamax',
    'ASGD',
    'NAdam',
    'RAdam',
    'RMSprop',
    'Rprop',
    'SGD'],
    'type': 'factor',
    'default': 'SGD',
    'transform': 'None',
    'class_name': 'torch.optim',
    'core_model_parameter_type': 'str',
    'lower': 0,
    'upper': 11},
'dropout_prob': {'type': 'float',
    'default': 0.01,
    'transform': 'None',
    'lower': 0.0,
    'upper': 0.25},
'lr_mult': {'type': 'float',
    'default': 1.0,
    'transform': 'None',
    'lower': 0.1,
    'upper': 10.0},
'patience': {'type': 'int',
    'default': 2,
    'transform': 'transform_power_2_int',
    'lower': 2,
    'upper': 6},
'initialization': {'levels': ['Default', 'Kaiming', 'Xavier'],
    'type': 'factor',
    'default': 'Default',
    'transform': 'None',
    'core_model_parameter_type': 'str',
    'lower': 0,
    'upper': 2}}

```

The NetLightBase is a configurable neural network. The hyperparameters of the model are

specified in the `core_model_hyper_dict` dictionary [\[SOURCE\]](#).

21.6 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model`

`spotPython` provides functions for modifying the hyperparameters, their bounds and factors as well as for activating and de-activating hyperparameters without re-compilation of the Python source code. These functions were described in [Section 14.6](#).

 **Caution:** Small number of epochs for demonstration purposes

- `epochs` and `patience` are set to small values for demonstration purposes. These values are too small for a real application.
- More resonable values are, e.g.:
 - `fun_control = modify_hyper_parameter_bounds(fun_control, "epochs", bounds=[7, 9])` and
 - `fun_control = modify_hyper_parameter_bounds(fun_control, "patience", bounds=[2, 7])`

```
from spotPython.hyperparameters.values import modify_hyper_parameter_bounds
```

```
fun_control = modify_hyper_parameter_bounds(fun_control, "l1", bounds=[6,13])
fun_control = modify_hyper_parameter_bounds(fun_control, "epochs", bounds=[6,13])
fun_control = modify_hyper_parameter_bounds(fun_control, "batch_size", bounds=[2, 8])
```

```
from spotPython.hyperparameters.values import modify_hyper_parameter_levels
```

```
fun_control = modify_hyper_parameter_levels(fun_control, "optimizer", ["Adam", "AdamW", "Ad
# fun_control = modify_hyper_parameter_levels(fun_control, "optimizer", ["Adam"])
```

The updated `fun_control` dictionary is shown below.

```
fun_control["core_model_hyper_dict"]
```

```
{'l1': {'type': 'int',
'default': 3,
'transform': 'transform_power_2_int',
'lower': 6,
'upper': 13},
'epochs': {'type': 'int',
```

```

'default': 4,
'transform': 'transform_power_2_int',
'lower': 6,
'upper': 13},
'batch_size': {'type': 'int',
'default': 4,
'transform': 'transform_power_2_int',
'lower': 2,
'upper': 8},
'act_fn': {'levels': ['Sigmoid', 'Tanh', 'ReLU', 'LeakyReLU', 'ELU', 'Swish'],
'type': 'factor',
'default': 'ReLU',
'transform': 'None',
'class_name': 'spotPython.torch.activation',
'core_model_parameter_type': 'instance()',
'lower': 0,
'upper': 5},
'optimizer': {'levels': ['Adam', 'AdamW', 'Adamax', 'NAdam'],
'type': 'factor',
'default': 'SGD',
'transform': 'None',
'class_name': 'torch.optim',
'core_model_parameter_type': 'str',
'lower': 0,
'upper': 3},
'dropout_prob': {'type': 'float',
'default': 0.01,
'transform': 'None',
'lower': 0.0,
'upper': 0.25},
'lr_mult': {'type': 'float',
'default': 1.0,
'transform': 'None',
'lower': 0.1,
'upper': 10.0},
'patience': {'type': 'int',
'default': 2,
'transform': 'transform_power_2_int',
'lower': 2,
'upper': 6},
'initialization': {'levels': ['Default', 'Kaiming', 'Xavier'],
'type': 'factor',
'default': 'Default',

```

```
'transform': 'None',
'core_model_parameter_type': 'str',
'lower': 0,
'upper': 2}}
```

21.7 Step 7: Data Splitting, the Objective (Loss) Function and the Metric

21.7.1 Evaluation

The evaluation procedure requires the specification of two elements:

1. the way how the data is split into a train and a test set (see Section [14.7.1](#))
2. the loss function (and a metric).

Caution: Data Splitting in Lightning

- The data splitting is handled by **Lightning**.

21.7.2 Loss Functions and Metrics

The loss function is specified in the configurable network class [\[SOURCE\]](#) We will use CrossEntropy loss for the multiclass-classification task.

21.7.3 Metric

- We will use the MAP@k metric [\[SOURCE\]](#) for the evaluation of the model. Here is an example how this metric is calculated.

```
from spotPython.torch.mapk import MAPK
import torch
mapk = MAPK(k=2)
target = torch.tensor([0, 1, 2, 2])
preds = torch.tensor(
    [
        [0.5, 0.2, 0.2], # 0 is in top 2
        [0.3, 0.4, 0.2], # 1 is in top 2
        [0.2, 0.4, 0.3], # 2 is in top 2
        [0.7, 0.2, 0.1], # 2 isn't in top 2
    ]
)
```

```

    ]
)
mapk.update(preds, target)
print(mapk.compute()) # tensor(0.6250)

```

tensor(0.6250)

Similar to the loss function, the metric is specified in the configurable network class [\[SOURCE\]](#).

Caution: Loss Function and Metric in Lightning

- The loss function and the metric are not hyperparameters that can be tuned with `spotPython`.
- They are handled by `Lightning`.

21.8 Step 8: Calling the SPOT Function

21.8.1 Preparing the SPOT Call

The following code passes the information about the parameter ranges and bounds to `spot`. It extracts the variable types, names, and bounds

```

from spotPython.hyperparameters.values import (get_bound_values,
        get_var_name,
        get_var_type,)
var_type = get_var_type(fun_control)
var_name = get_var_name(fun_control)
fun_control.update({"var_type": var_type,
                   "var_name": var_name})
lower = get_bound_values(fun_control, "lower")
upper = get_bound_values(fun_control, "upper")

```

Now, the dictionary `fun_control` contains all information needed for the hyperparameter tuning. Before the hyperparameter tuning is started, it is recommended to take a look at the experimental design. The method `gen_design_table` [\[SOURCE\]](#) generates a design table as follows:

```
from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control))
```

name	type	default	lower	upper	transform
l1	int	3	6	13	transform_power_2_int
epochs	int	4	6	13	transform_power_2_int
batch_size	int	4	2	8	transform_power_2_int
act_fn	factor	ReLU	0	5	None
optimizer	factor	SGD	0	3	None
dropout_prob	float	0.01	0	0.25	None
lr_mult	float	1.0	0.1	10	None
patience	int	2	2	6	transform_power_2_int
initialization	factor	Default	0	2	None

This allows to check if all information is available and if the information is correct.

21.8.2 The Objective Function fun

The objective function `fun` from the class `HyperLight` [\[SOURCE\]](#) is selected next. It implements an interface from PyTorch's training, validation, and testing methods to `spotPython`.

```
from spotPython.light.hyperlight import HyperLight
fun = HyperLight().fun
```

```
fun_control
```

```
{'CHECKPOINT_PATH': 'saved_models/',
 'DATASET_PATH': 'data/',
 'RESULTS_PATH': 'results/',
 'TENSORBOARD_PATH': 'runs/',
 '_L_in': 64,
 '_L_out': 11,
 'data': None,
 'data_dir': './data',
 'device': 'cpu',
 'enable_progress_bar': False,
 'eval': None,
 'k_folds': None,
```

```

'loss_function': None,
'metric_river': None,
'metric_sklearn': None,
'metric_torch': None,
'metric_params': {},
'model_dict': {},
'n_samples': None,
'num_workers': 0,
'optimizer': None,
'path': None,
'prep_model': None,
'save_model': False,
'show_batch_interval': 1000000,
'shuffle': None,
'target_column': None,
'train': None,
'test': None,
'task': 'classification',
'tensorboard_path': None,
'weights': 1.0,
'writer': None,
'core_model': spotPython.light.netlightbase.NetLightBase,
'core_model_hyper_dict': {'l1': {'type': 'int',
    'default': 3,
    'transform': 'transform_power_2_int',
    'lower': 6,
    'upper': 13},
    'epochs': {'type': 'int',
    'default': 4,
    'transform': 'transform_power_2_int',
    'lower': 6,
    'upper': 13},
    'batch_size': {'type': 'int',
    'default': 4,
    'transform': 'transform_power_2_int',
    'lower': 2,
    'upper': 8},
    'act_fn': {'levels': ['Sigmoid',
    'Tanh',
    'ReLU',
    'LeakyReLU',
    'ELU',
    'Swish']},

```

```

    'type': 'factor',
    'default': 'ReLU',
    'transform': 'None',
    'class_name': 'spotPython.torch.activation',
    'core_model_parameter_type': 'instance()',
    'lower': 0,
    'upper': 5},
    'optimizer': {'levels': ['Adam', 'AdamW', 'Adamax', 'NAdam'],
    'type': 'factor',
    'default': 'SGD',
    'transform': 'None',
    'class_name': 'torch.optim',
    'core_model_parameter_type': 'str',
    'lower': 0,
    'upper': 3},
    'dropout_prob': {'type': 'float',
    'default': 0.01,
    'transform': 'None',
    'lower': 0.0,
    'upper': 0.25},
    'lr_mult': {'type': 'float',
    'default': 1.0,
    'transform': 'None',
    'lower': 0.1,
    'upper': 10.0},
    'patience': {'type': 'int',
    'default': 2,
    'transform': 'transform_power_2_int',
    'lower': 2,
    'upper': 6},
    'initialization': {'levels': ['Default', 'Kaiming', 'Xavier'],
    'type': 'factor',
    'default': 'Default',
    'transform': 'None',
    'core_model_parameter_type': 'str',
    'lower': 0,
    'upper': 2}},
    'var_type': ['int',
    'int',
    'int',
    'factor',
    'factor',
    'float',

```



```

'float',
'int',
'factor'],
'var_name': ['l1',
'epochs',
'batch_size',
'act_fn',
'optimizer',
'dropout_prob',
'lr_mult',
'patience',
'initialization']]

```

21.8.3 Starting the Hyperparameter Tuning

The `spotPython` hyperparameter tuning is started by calling the `Spot` function [\[SOURCE\]](#) as described in Section [14.8.4](#).

```

import numpy as np
from spotPython.spot import spot
from math import inf
spot_tuner = spot.Spot(fun=fun,
                        lower = lower,
                        upper = upper,
                        fun_evals = inf,
                        fun_repeats = 1,
                        max_time = MAX_TIME,
                        noise = False,
                        tolerance_x = np.sqrt(np.spacing(1)),
                        var_type = var_type,
                        var_name = var_name,
                        infill_criterion = "y",
                        n_points = 1,
                        seed=123,
                        log_level = 50,
                        show_models= False,
                        show_progress= True,
                        fun_control = fun_control,
                        design_control={"init_size": INIT_SIZE,
                                      "repeats": 1},
                        surrogate_control={"noise": True,

```

```

        "cod_type": "norm",
        "min_theta": -4,
        "max_theta": 3,
        "n_theta": len(var_name),
        "model_fun_evals": 10_000,
        "log_level": 50
    })

spot_tuner.run()

```

```

config: {'l1': 4096, 'epochs': 4096, 'batch_size': 32, 'act_fn': ReLU(), 'optimizer': 'AdamW',
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=4096, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.10939527466721133, inplace=False)
    (3): Linear(in_features=4096, out_features=2048, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.10939527466721133, inplace=False)
    (6): Linear(in_features=2048, out_features=2048, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.10939527466721133, inplace=False)
    (9): Linear(in_features=2048, out_features=1024, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.10939527466721133, inplace=False)
    (12): Linear(in_features=1024, out_features=11, bias=True)
  )
)

```

Validate metric	DataLoader 0
hp_metric	2.433500051498413
val_acc	0.10954063385725021
val_loss	2.433500051498413
valid_mapk	0.1800411492586136

```

train_model result: {'valid_mapk': 0.1800411492586136, 'val_loss': 2.433500051498413, 'val_a

config: {'l1': 64, 'epochs': 128, 'batch_size': 256, 'act_fn': LeakyReLU(), 'optimizer': 'Ada
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): LeakyReLU()
    (2): Dropout(p=0.012926647388264517, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): LeakyReLU()
    (5): Dropout(p=0.012926647388264517, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): LeakyReLU()
    (8): Dropout(p=0.012926647388264517, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): LeakyReLU()
    (11): Dropout(p=0.012926647388264517, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)

```

Validate metric	DataLoader 0
hp_metric	2.238137722015381
val_acc	0.2862190902233124
val_loss	2.238137722015381
valid_mapk	0.45666956901550293

```

train_model result: {'valid_mapk': 0.45666956901550293, 'val_loss': 2.238137722015381, 'val_a

config: {'l1': 1024, 'epochs': 256, 'batch_size': 8, 'act_fn': Swish(), 'optimizer': 'NAdam'
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()

```

```

(valid_mapk): MAPK()
(test_mapk): MAPK()
(layers): Sequential(
  (0): Linear(in_features=64, out_features=1024, bias=True)
  (1): Swish()
  (2): Dropout(p=0.22086376796923401, inplace=False)
  (3): Linear(in_features=1024, out_features=512, bias=True)
  (4): Swish()
  (5): Dropout(p=0.22086376796923401, inplace=False)
  (6): Linear(in_features=512, out_features=512, bias=True)
  (7): Swish()
  (8): Dropout(p=0.22086376796923401, inplace=False)
  (9): Linear(in_features=512, out_features=256, bias=True)
  (10): Swish()
  (11): Dropout(p=0.22086376796923401, inplace=False)
  (12): Linear(in_features=256, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	2.4405665397644043
val_acc	0.10247349739074707
val_loss	2.4405665397644043
valid_mapk	0.18518517911434174

train_model result: {'valid_mapk': 0.18518517911434174, 'val_loss': 2.4405665397644043, 'val

config: {'l1': 512, 'epochs': 512, 'batch_size': 16, 'act_fn': Sigmoid(), 'optimizer': 'Adam

_L_in: 64

_L_out: 11

model: NetLightBase(

(train_mapk): MAPK()

(valid_mapk): MAPK()

(test_mapk): MAPK()

(layers): Sequential(

(0): Linear(in_features=64, out_features=512, bias=True)

(1): Sigmoid()

(2): Dropout(p=0.1890928563375006, inplace=False)

(3): Linear(in_features=512, out_features=256, bias=True)

```

(4): Sigmoid()
(5): Dropout(p=0.1890928563375006, inplace=False)
(6): Linear(in_features=256, out_features=256, bias=True)
(7): Sigmoid()
(8): Dropout(p=0.1890928563375006, inplace=False)
(9): Linear(in_features=256, out_features=128, bias=True)
(10): Sigmoid()
(11): Dropout(p=0.1890928563375006, inplace=False)
(12): Linear(in_features=128, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	2.3434996604919434
val_acc	0.19434629380702972
val_loss	2.3434996604919434
valid_mapk	0.2698337733745575

train_model result: {'valid_mapk': 0.2698337733745575, 'val_loss': 2.3434996604919434, 'val_

config: {'l1': 256, 'epochs': 4096, 'batch_size': 64, 'act_fn': ReLU(), 'optimizer': 'Adamax

_L_in: 64

_L_out: 11

```

model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=256, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.0708380794924471, inplace=False)
    (3): Linear(in_features=256, out_features=128, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.0708380794924471, inplace=False)
    (6): Linear(in_features=128, out_features=128, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.0708380794924471, inplace=False)
    (9): Linear(in_features=128, out_features=64, bias=True)
    (10): ReLU()
  )
)

```

```

(11): Dropout(p=0.0708380794924471, inplace=False)
(12): Linear(in_features=64, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	2.292955160140991
val_acc	0.2473498284816742
val_loss	2.292955160140991
valid_mapk	0.3427276313304901

```

train_model result: {'valid_mapk': 0.3427276313304901, 'val_loss': 2.292955160140991, 'val_a

```

```

config: {'l1': 64, 'epochs': 64, 'batch_size': 256, 'act_fn': LeakyReLU(), 'optimizer': 'Adam'
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): LeakyReLU()
    (2): Dropout(p=0.006064053615858084, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): LeakyReLU()
    (5): Dropout(p=0.006064053615858084, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): LeakyReLU()
    (8): Dropout(p=0.006064053615858084, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): LeakyReLU()
    (11): Dropout(p=0.006064053615858084, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)
)

```

Validate metric	DataLoader 0
hp_metric	2.3466885089874268
val_acc	0.208480566740036
val_loss	2.3466885089874268
valid_mapk	0.3177203834056854

train_model result: {'valid_mapk': 0.3177203834056854, 'val_loss': 2.3466885089874268, 'val_

spotPython tuning: 2.238137722015381 [-----] 3.17%

```

config: {'l1': 64, 'epochs': 512, 'batch_size': 256, 'act_fn': Sigmoid(), 'optimizer': 'Adam
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): Sigmoid()
    (2): Dropout(p=0.030100669185271465, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): Sigmoid()
    (5): Dropout(p=0.030100669185271465, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): Sigmoid()
    (8): Dropout(p=0.030100669185271465, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): Sigmoid()
    (11): Dropout(p=0.030100669185271465, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)

```

Validate metric	DataLoader 0
-----------------	--------------

hp_metric	2.298098564147949
val_acc	0.22968198359012604
val_loss	2.298098564147949
valid_mapk	0.2535686790943146

train_model result: {'valid_mapk': 0.2535686790943146, 'val_loss': 2.298098564147949, 'val_a

spotPython tuning: 2.238137722015381 [#-----] 7.31%

config: {'l1': 64, 'epochs': 128, 'batch_size': 256, 'act_fn': ReLU(), 'optimizer': 'NAdam',

_L_in: 64

_L_out: 11

```
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.03832172101534319, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.03832172101534319, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.03832172101534319, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.03832172101534319, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)
```

Validate metric

DataLoader 0

hp_metric

2.271381378173828

val_acc	0.2614840865135193
val_loss	2.271381378173828
valid_mapk	0.402874231338501

train_model result: {'valid_mapk': 0.402874231338501, 'val_loss': 2.271381378173828, 'val_acc': 0.2614840865135193}

spotPython tuning: 2.238137722015381 [#-----] 10.68%

config: {'l1': 64, 'epochs': 128, 'batch_size': 128, 'act_fn': Swish(), 'optimizer': 'AdamW'}

_L_in: 64

_L_out: 11

```
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): Swish()
    (2): Dropout(p=0.03644111923614535, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): Swish()
    (5): Dropout(p=0.03644111923614535, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): Swish()
    (8): Dropout(p=0.03644111923614535, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): Swish()
    (11): Dropout(p=0.03644111923614535, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)
```

Validate metric

DataLoader 0

hp_metric	2.278205156326294
val_acc	0.2473498284816742

val_loss	2.278205156326294
valid_mapk	0.3325938880443573

train_model result: {'valid_mapk': 0.3325938880443573, 'val_loss': 2.278205156326294, 'val_a

spotPython tuning: 2.238137722015381 [#-----] 14.89%

```

config: {'l1': 256, 'epochs': 256, 'batch_size': 8, 'act_fn': ReLU(), 'optimizer': 'Adam', 'o
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=256, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.07083807526728049, inplace=False)
    (3): Linear(in_features=256, out_features=128, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.07083807526728049, inplace=False)
    (6): Linear(in_features=128, out_features=128, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.07083807526728049, inplace=False)
    (9): Linear(in_features=128, out_features=64, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.07083807526728049, inplace=False)
    (12): Linear(in_features=64, out_features=11, bias=True)
  )
)

```

Validate metric	DataLoader 0
hp_metric	2.394630193710327
val_acc	0.1484098881483078
val_loss	2.394630193710327
valid_mapk	0.22974535822868347

train_model result: {'valid_mapk': 0.22974535822868347, 'val_loss': 2.394630193710327, 'val_acc': 0.2720848023891449}

spotPython tuning: 2.238137722015381 [##-----] 21.17%

```
config: {'l1': 64, 'epochs': 64, 'batch_size': 256, 'act_fn': Tanh(), 'optimizer': 'Adamax',
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.0, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.0, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.0, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.0, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)
```

Validate metric	DataLoader 0
hp_metric	2.25821590423584
val_acc	0.2720848023891449
val_loss	2.25821590423584
valid_mapk	0.3440875709056854

train_model result: {'valid_mapk': 0.3440875709056854, 'val_loss': 2.25821590423584, 'val_acc': 0.2720848023891449}

spotPython tuning: 2.238137722015381 [###-----] 26.55%

```

config: {'l1': 64, 'epochs': 128, 'batch_size': 128, 'act_fn': LeakyReLU(), 'optimizer': 'Ada
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): LeakyReLU()
    (2): Dropout(p=0.007092367088098504, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): LeakyReLU()
    (5): Dropout(p=0.007092367088098504, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): LeakyReLU()
    (8): Dropout(p=0.007092367088098504, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): LeakyReLU()
    (11): Dropout(p=0.007092367088098504, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)

```

Validate metric	DataLoader 0
hp_metric	2.3103139400482178
val_acc	0.22968198359012604
val_loss	2.3103139400482178
valid_mapk	0.2670235335826874

```

train_model result: {'valid_mapk': 0.2670235335826874, 'val_loss': 2.3103139400482178, 'val_a

```

```

spotPython tuning: 2.238137722015381 [###-----] 34.01%

```

```

config: {'l1': 128, 'epochs': 128, 'batch_size': 256, 'act_fn': ReLU(), 'optimizer': 'AdamW'
_L_in: 64

```

```

_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=128, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.16965823251206952, inplace=False)
    (3): Linear(in_features=128, out_features=64, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.16965823251206952, inplace=False)
    (6): Linear(in_features=64, out_features=64, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.16965823251206952, inplace=False)
    (9): Linear(in_features=64, out_features=32, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.16965823251206952, inplace=False)
    (12): Linear(in_features=32, out_features=11, bias=True)
  )
)

```

Validate metric	DataLoader 0
hp_metric	2.2878034114837646
val_acc	0.24381625652313232
val_loss	2.2878034114837646
valid_mapk	0.3672960102558136

train_model result: {'valid_mapk': 0.3672960102558136, 'val_loss': 2.2878034114837646, 'val_

spotPython tuning: 2.238137722015381 [####-----] 36.82%

```

config: {'l1': 128, 'epochs': 64, 'batch_size': 256, 'act_fn': LeakyReLU(), 'optimizer': 'Ad
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()

```

```

(valid_mapk): MAPK()
(test_mapk): MAPK()
(layers): Sequential(
  (0): Linear(in_features=64, out_features=128, bias=True)
  (1): LeakyReLU()
  (2): Dropout(p=0.13070796890136097, inplace=False)
  (3): Linear(in_features=128, out_features=64, bias=True)
  (4): LeakyReLU()
  (5): Dropout(p=0.13070796890136097, inplace=False)
  (6): Linear(in_features=64, out_features=64, bias=True)
  (7): LeakyReLU()
  (8): Dropout(p=0.13070796890136097, inplace=False)
  (9): Linear(in_features=64, out_features=32, bias=True)
  (10): LeakyReLU()
  (11): Dropout(p=0.13070796890136097, inplace=False)
  (12): Linear(in_features=32, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	2.272099018096924
val_acc	0.2614840865135193
val_loss	2.272099018096924
valid_mapk	0.3734809160232544

train_model result: {'valid_mapk': 0.3734809160232544, 'val_loss': 2.272099018096924, 'val_a

spotPython tuning: 2.238137722015381 [####-----] 41.20%

```

config: {'l1': 64, 'epochs': 128, 'batch_size': 256, 'act_fn': Swish(), 'optimizer': 'Adamax',
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(

```

```

(0): Linear(in_features=64, out_features=64, bias=True)
(1): Swish()
(2): Dropout(p=0.08985395193916346, inplace=False)
(3): Linear(in_features=64, out_features=32, bias=True)
(4): Swish()
(5): Dropout(p=0.08985395193916346, inplace=False)
(6): Linear(in_features=32, out_features=32, bias=True)
(7): Swish()
(8): Dropout(p=0.08985395193916346, inplace=False)
(9): Linear(in_features=32, out_features=16, bias=True)
(10): Swish()
(11): Dropout(p=0.08985395193916346, inplace=False)
(12): Linear(in_features=16, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	2.2256393432617188
val_acc	0.31448763608932495
val_loss	2.2256393432617188
valid_mapk	0.3494887948036194

train_model result: {'valid_mapk': 0.3494887948036194, 'val_loss': 2.2256393432617188, 'val_

spotPython tuning: 2.2256393432617188 [#####-----] 46.62%

```

config: {'l1': 64, 'epochs': 1024, 'batch_size': 256, 'act_fn': ReLU(), 'optimizer': 'NAdam'
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.0, inplace=False)

```

```

(3): Linear(in_features=64, out_features=32, bias=True)
(4): ReLU()
(5): Dropout(p=0.0, inplace=False)
(6): Linear(in_features=32, out_features=32, bias=True)
(7): ReLU()
(8): Dropout(p=0.0, inplace=False)
(9): Linear(in_features=32, out_features=16, bias=True)
(10): ReLU()
(11): Dropout(p=0.0, inplace=False)
(12): Linear(in_features=16, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	2.3017842769622803
val_acc	0.22968198359012604
val_loss	2.3017842769622803
valid_mapk	0.2447916567325592

train_model result: {'valid_mapk': 0.2447916567325592, 'val_loss': 2.3017842769622803, 'val_a

spotPython tuning: 2.2256393432617188 [#####----] 56.74%

```

config: {'l1': 64, 'epochs': 128, 'batch_size': 256, 'act_fn': LeakyReLU(), 'optimizer': 'Ada
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): LeakyReLU()
    (2): Dropout(p=0.001989156451688859, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): LeakyReLU()
    (5): Dropout(p=0.001989156451688859, inplace=False)

```



```

(6): Linear(in_features=32, out_features=32, bias=True)
(7): LeakyReLU()
(8): Dropout(p=0.001989156451688859, inplace=False)
(9): Linear(in_features=32, out_features=16, bias=True)
(10): LeakyReLU()
(11): Dropout(p=0.001989156451688859, inplace=False)
(12): Linear(in_features=16, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	2.378943681716919
val_acc	0.1236749142408371
val_loss	2.378943681716919
valid_mapk	0.22030526399612427

train_model result: {'valid_mapk': 0.22030526399612427, 'val_loss': 2.378943681716919, 'val_

spotPython tuning: 2.2256393432617188 [#####----] 62.46%

```

config: {'l1': 64, 'epochs': 8192, 'batch_size': 256, 'act_fn': ELU(), 'optimizer': 'AdamW',
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): ELU()
    (2): Dropout(p=0.06870358263506916, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): ELU()
    (5): Dropout(p=0.06870358263506916, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): ELU()
    (8): Dropout(p=0.06870358263506916, inplace=False)

```

```

(9): Linear(in_features=32, out_features=16, bias=True)
(10): ELU()
(11): Dropout(p=0.06870358263506916, inplace=False)
(12): Linear(in_features=16, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	2.2920384407043457
val_acc	0.24028268456459045
val_loss	2.2920384407043457
valid_mapk	0.3317177891731262

train_model result: {'valid_mapk': 0.3317177891731262, 'val_loss': 2.2920384407043457, 'val_

spotPython tuning: 2.2256393432617188 [#####---] 67.01%

```

config: {'l1': 64, 'epochs': 512, 'batch_size': 256, 'act_fn': LeakyReLU(), 'optimizer': 'Ad
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): LeakyReLU()
    (2): Dropout(p=0.08103738523239842, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): LeakyReLU()
    (5): Dropout(p=0.08103738523239842, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): LeakyReLU()
    (8): Dropout(p=0.08103738523239842, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): LeakyReLU()
    (11): Dropout(p=0.08103738523239842, inplace=False)
  )
)

```

```

    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)

```

Validate metric	DataLoader 0
hp_metric	2.292820453643799
val_acc	0.2473498284816742
val_loss	2.292820453643799
valid_mapk	0.30070891976356506

train_model result: {'valid_mapk': 0.30070891976356506, 'val_loss': 2.292820453643799, 'val_

spotPython tuning: 2.2256393432617188 [#####---] 73.60%

config: {'l1': 128, 'epochs': 512, 'batch_size': 256, 'act_fn': Tanh(), 'optimizer': 'NAdam',
 _L_in: 64
 _L_out: 11

```

model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=128, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.14105555315807383, inplace=False)
    (3): Linear(in_features=128, out_features=64, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.14105555315807383, inplace=False)
    (6): Linear(in_features=64, out_features=64, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.14105555315807383, inplace=False)
    (9): Linear(in_features=64, out_features=32, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.14105555315807383, inplace=False)
    (12): Linear(in_features=32, out_features=11, bias=True)
  )
)

```

Validate metric	DataLoader 0
hp_metric	2.245398759841919
val_acc	0.2968197762966156
val_loss	2.245398759841919
valid_mapk	0.45798367261886597

train_model result: {'valid_mapk': 0.45798367261886597, 'val_loss': 2.245398759841919, 'val_

spotPython tuning: 2.2256393432617188 [#####--] 80.39%

```

config: {'l1': 128, 'epochs': 4096, 'batch_size': 256, 'act_fn': Sigmoid(), 'optimizer': 'NA
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=128, bias=True)
    (1): Sigmoid()
    (2): Dropout(p=0.14118067105660784, inplace=False)
    (3): Linear(in_features=128, out_features=64, bias=True)
    (4): Sigmoid()
    (5): Dropout(p=0.14118067105660784, inplace=False)
    (6): Linear(in_features=64, out_features=64, bias=True)
    (7): Sigmoid()
    (8): Dropout(p=0.14118067105660784, inplace=False)
    (9): Linear(in_features=64, out_features=32, bias=True)
    (10): Sigmoid()
    (11): Dropout(p=0.14118067105660784, inplace=False)
    (12): Linear(in_features=32, out_features=11, bias=True)
  )
)

```

Validate metric	DataLoader 0
-----------------	--------------

hp_metric	2.2562859058380127
val_acc	0.27561837434768677
val_loss	2.2562859058380127
valid_mapk	0.432086706161499

train_model result: {'valid_mapk': 0.432086706161499, 'val_loss': 2.2562859058380127, 'val_a

spotPython tuning: 2.2256393432617188 [#####-] 89.41%

config: {'l1': 256, 'epochs': 512, 'batch_size': 256, 'act_fn': Tanh(), 'optimizer': 'Adamax'
 _L_in: 64
 _L_out: 11

```
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=256, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.078737605816338, inplace=False)
    (3): Linear(in_features=256, out_features=128, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.078737605816338, inplace=False)
    (6): Linear(in_features=128, out_features=128, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.078737605816338, inplace=False)
    (9): Linear(in_features=128, out_features=64, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.078737605816338, inplace=False)
    (12): Linear(in_features=64, out_features=11, bias=True)
  )
)
```

Validate metric

DataLoader 0

hp_metric	2.257312059402466
val_acc	0.27561837434768677
val_loss	2.257312059402466

valid_mapk 0.4337504804134369

train_model result: {'valid_mapk': 0.4337504804134369, 'val_loss': 2.257312059402466, 'val_a

spotPython tuning: 2.2256393432617188 [#####] 95.14%

config: {'l1': 256, 'epochs': 64, 'batch_size': 256, 'act_fn': Swish(), 'optimizer': 'Adamax'
_L_in: 64

_L_out: 11

model: NetLightBase(
(train_mapk): MAPK()
(valid_mapk): MAPK()
(test_mapk): MAPK()
(layers): Sequential(
(0): Linear(in_features=64, out_features=256, bias=True)
(1): Swish()
(2): Dropout(p=0.0, inplace=False)
(3): Linear(in_features=256, out_features=128, bias=True)
(4): Swish()
(5): Dropout(p=0.0, inplace=False)
(6): Linear(in_features=128, out_features=128, bias=True)
(7): Swish()
(8): Dropout(p=0.0, inplace=False)
(9): Linear(in_features=128, out_features=64, bias=True)
(10): Swish()
(11): Dropout(p=0.0, inplace=False)
(12): Linear(in_features=64, out_features=11, bias=True)
)
)

Validate metric

DataLoader 0

hp_metric	2.2881009578704834
val_acc	0.22968198359012604
val_loss	2.2881009578704834
valid_mapk	0.30428963899612427

train_model result: {'valid_mapk': 0.30428963899612427, 'val_loss': 2.2881009578704834, 'val.

```
spotPython tuning: 2.2256393432617188 [#####] 100.00% Done...
```

```
<spotPython.spot.spot.Spot at 0x188dbbb20>
```

21.9 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section 14.9, see also the description in the documentation: [Tensorboard](#).

21.10 Step 10: Results

After the hyperparameter tuning run is finished, the results can be analyzed as described in Section 14.10.

```
spot_tuner.plot_progress(log_y=False,  
    filename="./figures/" + experiment_name+"_progress.png")
```

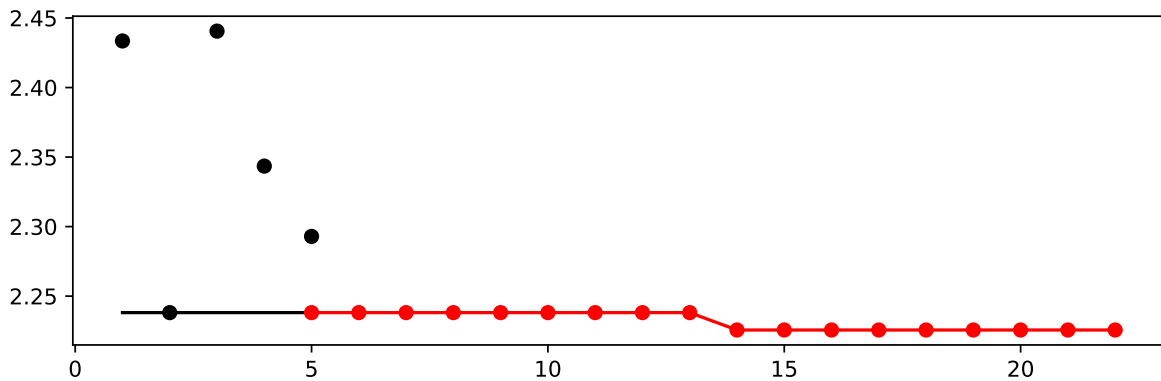


Figure 21.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

```
from spotPython.utils.eda import gen_design_table  
print(gen_design_table(fun_control=fun_control, spot=spot_tuner))
```

name	type	default	lower	upper	tuned	transform
11	int	3	6.0	13.0	6.0	transform_

epochs	int	4		6.0		13.0			7.0	transform_
batch_size	int	4		2.0		8.0			8.0	transform_
act_fn	factor	ReLU		0.0		5.0			5.0	None
optimizer	factor	SGD		0.0		3.0			2.0	None
dropout_prob	float	0.01		0.0		0.25		0.08985395193916346		None
lr_mult	float	1.0		0.1		10.0		2.1647103021174496		None
patience	int	2		2.0		6.0			4.0	transform_
initialization	factor	Default		0.0		2.0			1.0	None

```
spot_tuner.plot_importance(threshold=0.025,
                           filename="./figures/" + experiment_name+"_importance.png")
```

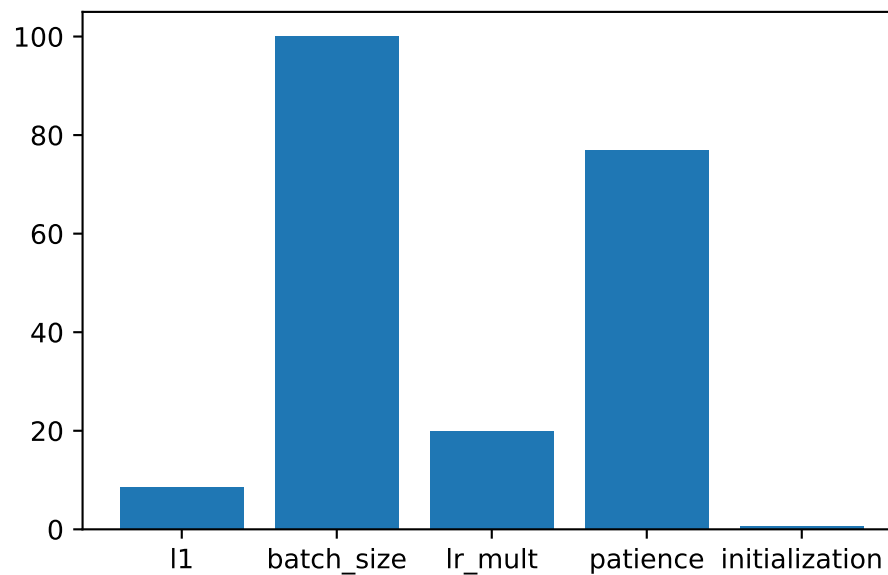


Figure 21.2: Variable importance plot, threshold 0.025.

21.10.1 Get the Tuned Architecture

```
from spotPython.light.utils import get_tuned_architecture
config = get_tuned_architecture(spot_tuner, fun_control)
```

- Test on the full data set

```
from spotPython.light.traintest import test_model
test_model(config, fun_control)
```



```

model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): Swish()
    (2): Dropout(p=0.08985395193916346, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): Swish()
    (5): Dropout(p=0.08985395193916346, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): Swish()
    (8): Dropout(p=0.08985395193916346, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): Swish()
    (11): Dropout(p=0.08985395193916346, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)

```

Test metric	DataLoader 0
hp_metric	2.184826135635376
test_mapk_epoch	0.42180153727531433
val_acc	0.3578500747680664
val_loss	2.184826135635376

test_model result: {'test_mapk_epoch': 0.42180153727531433, 'val_loss': 2.184826135635376, 'val_acc': 0.3578500747680664}

(2.184826135635376, 0.3578500747680664)

```

from spotPython.light.traintest import load_light_from_checkpoint

model_loaded = load_light_from_checkpoint(config, fun_control)

```

Loading model from runs/lightning_logs/64_128_256_Swish()_Adamax_0.08985395193916346_2.16471

21.10.2 Cross Validation With Lightning

- The `KFold` class from `sklearn.model_selection` is used to generate the folds for cross-validation.
- These mechanism is used to generate the folds for the final evaluation of the model.
- The `CrossValidationDataModule` class [\[SOURCE\]](#) is used to generate the folds for the hyperparameter tuning process.
- It is called from the `cv_model` function [\[SOURCE\]](#).

```
from spotPython.light.traintest import cv_model
cv_model(config, fun_control)
```

```
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): Swish()
    (2): Dropout(p=0.08985395193916346, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): Swish()
    (5): Dropout(p=0.08985395193916346, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): Swish()
    (8): Dropout(p=0.08985395193916346, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): Swish()
    (11): Dropout(p=0.08985395193916346, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)
k: 0
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): Swish()
    (2): Dropout(p=0.08985395193916346, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
```

```

(4): Swish()
(5): Dropout(p=0.08985395193916346, inplace=False)
(6): Linear(in_features=32, out_features=32, bias=True)
(7): Swish()
(8): Dropout(p=0.08985395193916346, inplace=False)
(9): Linear(in_features=32, out_features=16, bias=True)
(10): Swish()
(11): Dropout(p=0.08985395193916346, inplace=False)
(12): Linear(in_features=16, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	2.1515114307403564
val_acc	0.3661971688270569
val_loss	2.1515114307403564
valid_mapk	0.4624413251876831

train_model result: {'valid_mapk': 0.4624413251876831, 'val_loss': 2.1515114307403564, 'val_acc': 0.3661971688270569, 'hp_metric': 2.1515114307403564}

```

model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): Swish()
    (2): Dropout(p=0.08985395193916346, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): Swish()
    (5): Dropout(p=0.08985395193916346, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): Swish()
    (8): Dropout(p=0.08985395193916346, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): Swish()
    (11): Dropout(p=0.08985395193916346, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)

```

)

Validate metric	DataLoader 0
hp_metric	2.0143375396728516
val_acc	0.5633803009986877
val_loss	2.0143375396728516
valid_mapk	0.6314554214477539

train_model result: {'valid_mapk': 0.6314554214477539, 'val_loss': 2.0143375396728516, 'val_acc': 0.5633803009986877, 'val_mapk': 0.6314554214477539}

```
model: NetLightBase(  
  (train_mapk): MAPK()  
  (valid_mapk): MAPK()  
  (test_mapk): MAPK()  
  (layers): Sequential(  
    (0): Linear(in_features=64, out_features=64, bias=True)  
    (1): Swish()  
    (2): Dropout(p=0.08985395193916346, inplace=False)  
    (3): Linear(in_features=64, out_features=32, bias=True)  
    (4): Swish()  
    (5): Dropout(p=0.08985395193916346, inplace=False)  
    (6): Linear(in_features=32, out_features=32, bias=True)  
    (7): Swish()  
    (8): Dropout(p=0.08985395193916346, inplace=False)  
    (9): Linear(in_features=32, out_features=16, bias=True)  
    (10): Swish()  
    (11): Dropout(p=0.08985395193916346, inplace=False)  
    (12): Linear(in_features=16, out_features=11, bias=True)  
  )  
)
```

Validate metric	DataLoader 0
hp_metric	2.1380867958068848
val_acc	0.39436620473861694
val_loss	2.1380867958068848
valid_mapk	0.4530516564846039

```
train_model result: {'valid_mapk': 0.4530516564846039, 'val_loss': 2.1380867958068848, 'val_acc': 0.5492957830429077, 'val_mapk': 0.5985915660858154}
```

```
model: NetLightBase(  
  (train_mapk): MAPK()  
  (valid_mapk): MAPK()  
  (test_mapk): MAPK()  
  (layers): Sequential(  
    (0): Linear(in_features=64, out_features=64, bias=True)  
    (1): Swish()  
    (2): Dropout(p=0.08985395193916346, inplace=False)  
    (3): Linear(in_features=64, out_features=32, bias=True)  
    (4): Swish()  
    (5): Dropout(p=0.08985395193916346, inplace=False)  
    (6): Linear(in_features=32, out_features=32, bias=True)  
    (7): Swish()  
    (8): Dropout(p=0.08985395193916346, inplace=False)  
    (9): Linear(in_features=32, out_features=16, bias=True)  
    (10): Swish()  
    (11): Dropout(p=0.08985395193916346, inplace=False)  
    (12): Linear(in_features=16, out_features=11, bias=True)  
  )  
)
```

Validate metric	DataLoader 0
hp_metric	2.005397081375122
val_acc	0.5492957830429077
val_loss	2.005397081375122
valid_mapk	0.5985915660858154

```
train_model result: {'valid_mapk': 0.5985915660858154, 'val_loss': 2.005397081375122, 'val_acc': 0.5492957830429077, 'val_mapk': 0.5985915660858154}
```

```
model: NetLightBase(  
  (train_mapk): MAPK()  
  (valid_mapk): MAPK()  
  (test_mapk): MAPK()  
  (layers): Sequential(  
    (0): Linear(in_features=64, out_features=64, bias=True)  
    (1): Swish()  
    (2): Dropout(p=0.08985395193916346, inplace=False)
```

```

(3): Linear(in_features=64, out_features=32, bias=True)
(4): Swish()
(5): Dropout(p=0.08985395193916346, inplace=False)
(6): Linear(in_features=32, out_features=32, bias=True)
(7): Swish()
(8): Dropout(p=0.08985395193916346, inplace=False)
(9): Linear(in_features=32, out_features=16, bias=True)
(10): Swish()
(11): Dropout(p=0.08985395193916346, inplace=False)
(12): Linear(in_features=16, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	2.144458055496216
val_acc	0.39436620473861694
val_loss	2.144458055496216
valid_mapk	0.4553990662097931

train_model result: {'valid_mapk': 0.4553990662097931, 'val_loss': 2.144458055496216, 'val_acc': 0.39436620473861694}

```

model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): Swish()
    (2): Dropout(p=0.08985395193916346, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): Swish()
    (5): Dropout(p=0.08985395193916346, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): Swish()
    (8): Dropout(p=0.08985395193916346, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): Swish()
    (11): Dropout(p=0.08985395193916346, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)

```

)
)

Validate metric	DataLoader 0
hp_metric	2.033134937286377
val_acc	0.5211267471313477
val_loss	2.033134937286377
valid_mapk	0.6056337952613831

train_model result: {'valid_mapk': 0.6056337952613831, 'val_loss': 2.033134937286377, 'val_acc': 0.5211267471313477, 'valid_mapk': 0.6056337952613831}

```
model: NetLightBase(  
  (train_mapk): MAPK()  
  (valid_mapk): MAPK()  
  (test_mapk): MAPK()  
  (layers): Sequential(  
    (0): Linear(in_features=64, out_features=64, bias=True)  
    (1): Swish()  
    (2): Dropout(p=0.08985395193916346, inplace=False)  
    (3): Linear(in_features=64, out_features=32, bias=True)  
    (4): Swish()  
    (5): Dropout(p=0.08985395193916346, inplace=False)  
    (6): Linear(in_features=32, out_features=32, bias=True)  
    (7): Swish()  
    (8): Dropout(p=0.08985395193916346, inplace=False)  
    (9): Linear(in_features=32, out_features=16, bias=True)  
    (10): Swish()  
    (11): Dropout(p=0.08985395193916346, inplace=False)  
    (12): Linear(in_features=16, out_features=11, bias=True)  
  )  
)
```

Validate metric	DataLoader 0
hp_metric	2.017897129058838
val_acc	0.5492957830429077
val_loss	2.017897129058838
valid_mapk	0.6009389758110046

```
train_model result: {'valid_mapk': 0.6009389758110046, 'val_loss': 2.017897129058838, 'val_acc': 0.6009389758110046, 'val_mapk': 0.6009389758110046}
k: 7
```

```
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): Swish()
    (2): Dropout(p=0.08985395193916346, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): Swish()
    (5): Dropout(p=0.08985395193916346, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): Swish()
    (8): Dropout(p=0.08985395193916346, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): Swish()
    (11): Dropout(p=0.08985395193916346, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)
```

Validate metric	DataLoader 0
hp_metric	1.9059064388275146
val_acc	0.6428571343421936
val_loss	1.9059064388275146
valid_mapk	0.6928571462631226

```
train_model result: {'valid_mapk': 0.6928571462631226, 'val_loss': 1.9059064388275146, 'val_acc': 0.6428571343421936, 'val_mapk': 0.6928571462631226}
k: 8
```

```
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
```



```

(1): Swish()
(2): Dropout(p=0.08985395193916346, inplace=False)
(3): Linear(in_features=64, out_features=32, bias=True)
(4): Swish()
(5): Dropout(p=0.08985395193916346, inplace=False)
(6): Linear(in_features=32, out_features=32, bias=True)
(7): Swish()
(8): Dropout(p=0.08985395193916346, inplace=False)
(9): Linear(in_features=32, out_features=16, bias=True)
(10): Swish()
(11): Dropout(p=0.08985395193916346, inplace=False)
(12): Linear(in_features=16, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	1.9083349704742432
val_acc	0.6428571343421936
val_loss	1.9083349704742432
valid_mapk	0.6738095283508301

train_model result: {'valid_mapk': 0.6738095283508301, 'val_loss': 1.9083349704742432, 'val_acc': 0.6428571343421936, 'hp_metric': 1.9083349704742432}

```

model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): Swish()
    (2): Dropout(p=0.08985395193916346, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): Swish()
    (5): Dropout(p=0.08985395193916346, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): Swish()
    (8): Dropout(p=0.08985395193916346, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): Swish()
  )
)

```

```

(11): Dropout(p=0.08985395193916346, inplace=False)
(12): Linear(in_features=16, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	1.902524709701538
val_acc	0.6714285612106323
val_loss	1.902524709701538
valid_mapk	0.7214285731315613

train_model result: {'valid_mapk': 0.7214285731315613, 'val_loss': 1.902524709701538, 'val_a
cv_model mapk result: 0.5895607054233551

0.5895607054233551

i Note: Evaluation for the Final Comparison

- This is the evaluation that will be used in the comparison.

21.10.3 Detailed Hyperparameter Plots

```

filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)

```

```

l1: 8.624864607237388
batch_size: 100.0
lr_mult: 19.821240587050077
patience: 76.84773617766938
initialization: 0.7222436211219133

```

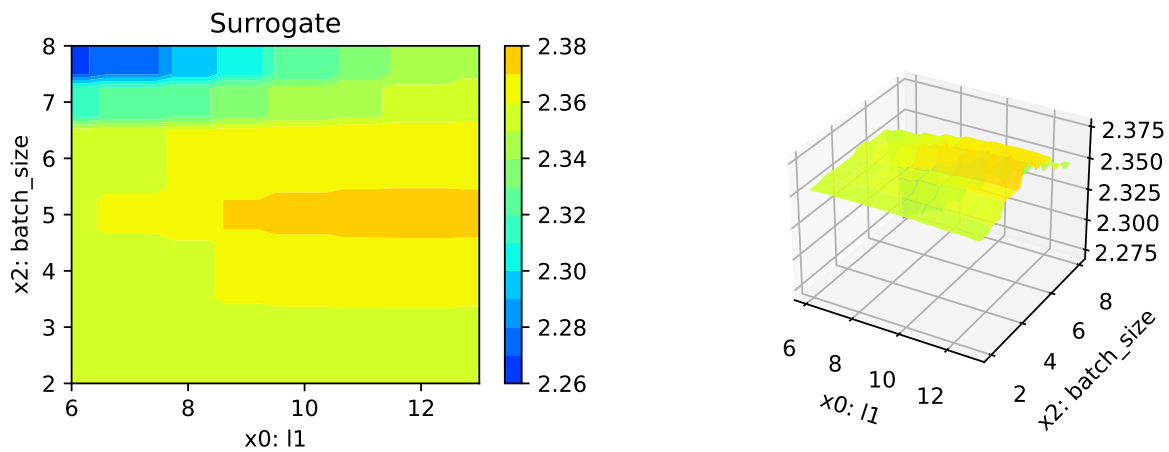
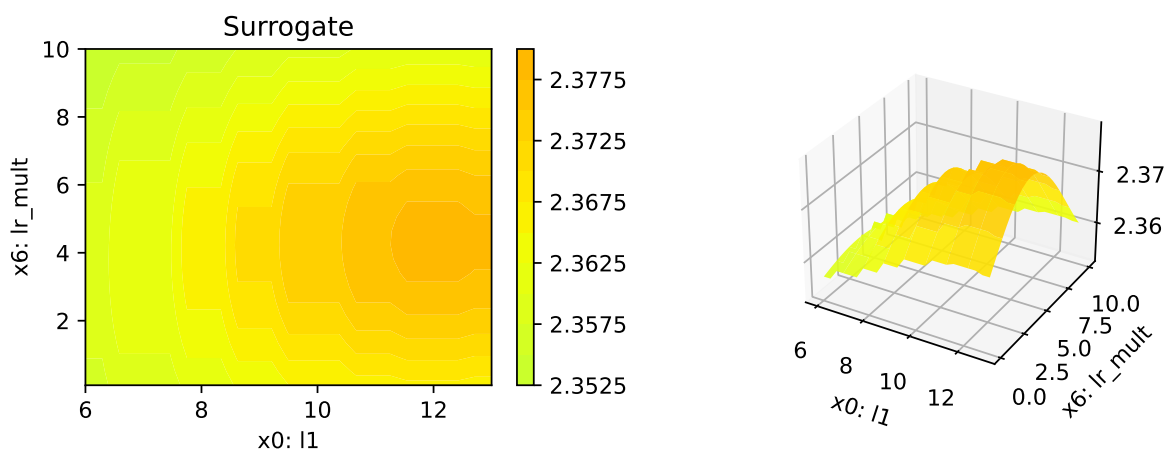
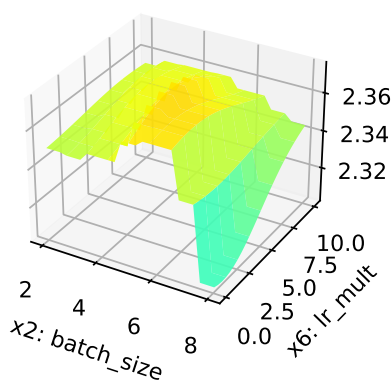
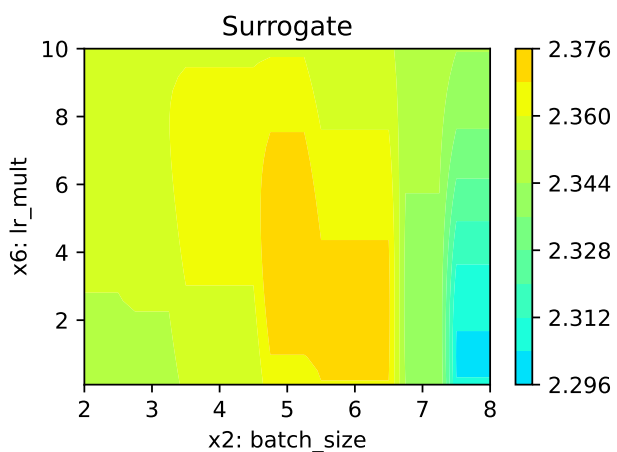
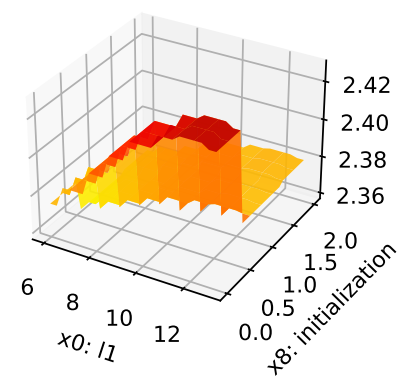
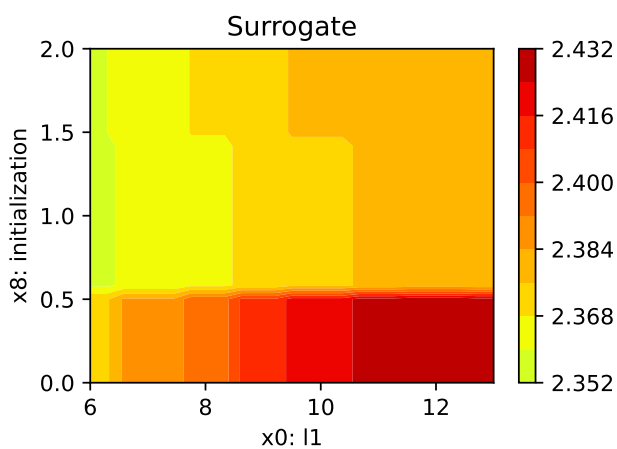
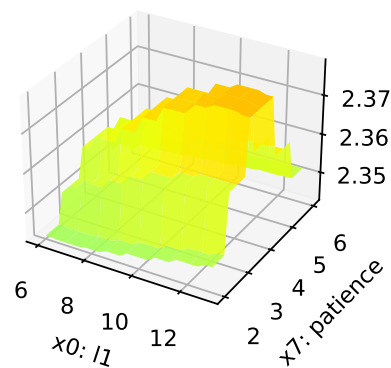
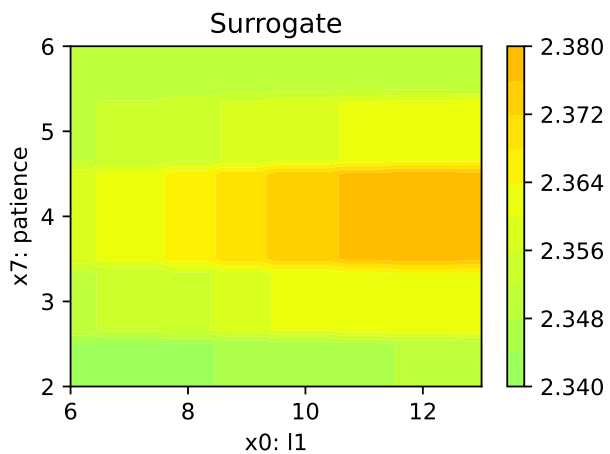
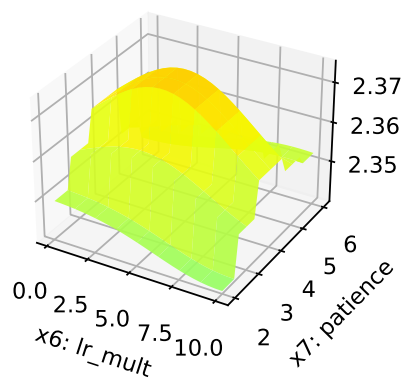
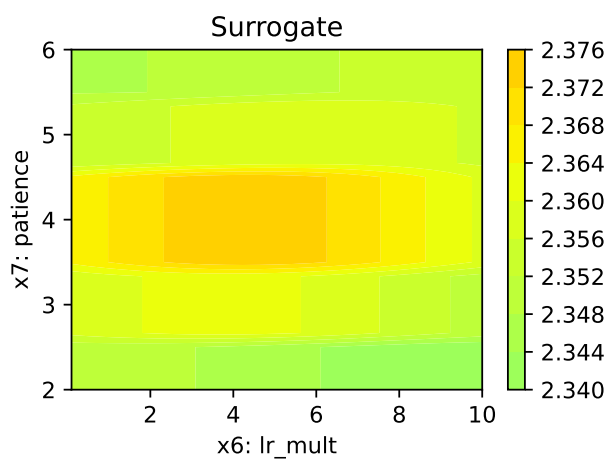
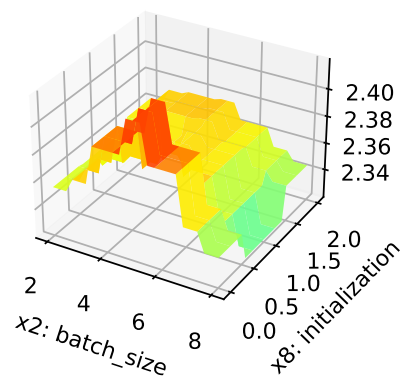
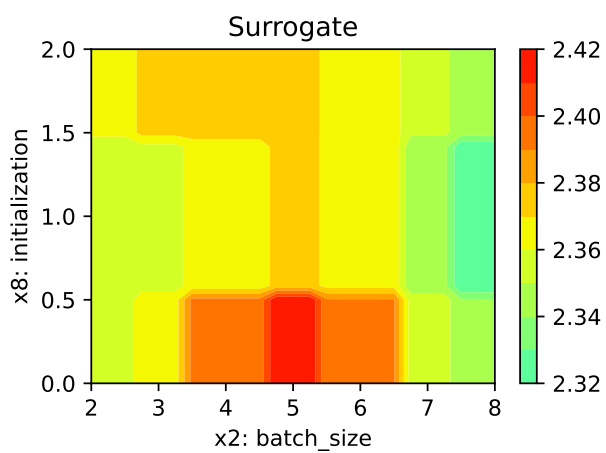
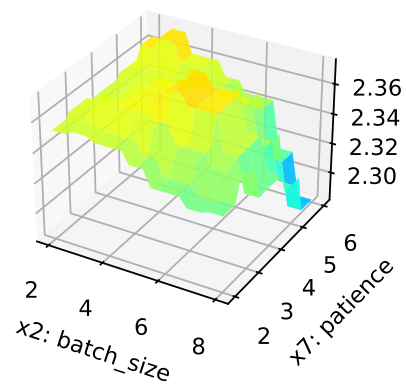
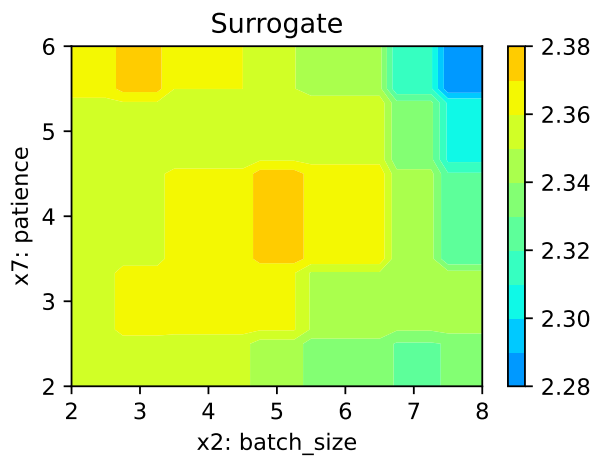
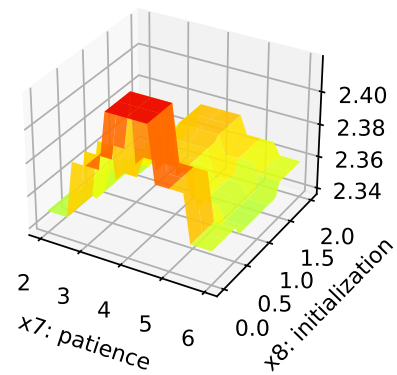
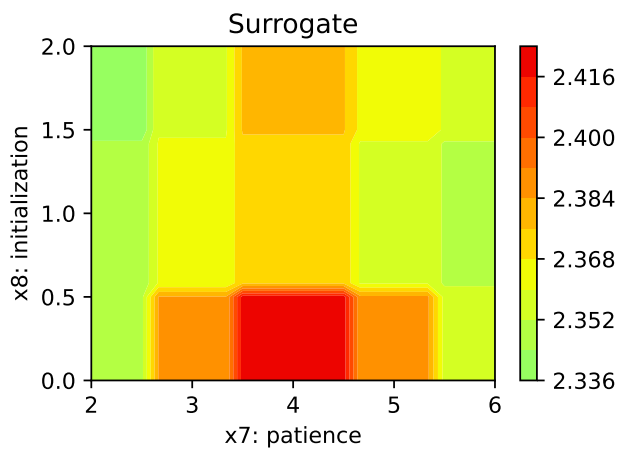
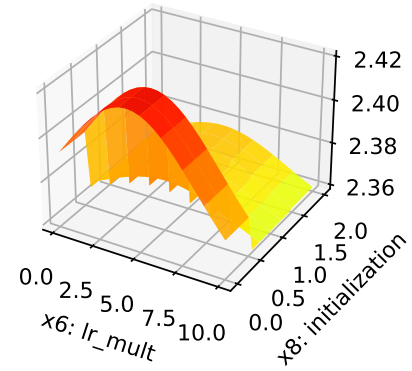
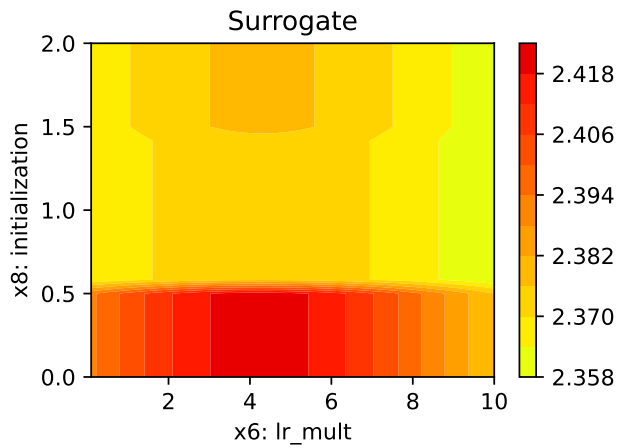


Figure 21.3: Contour plots.









21.10.4 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

Unable to display output for mime type(s): text/html

Parallel coordinates plots

Unable to display output for mime type(s): text/html

21.10.5 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

```
PLOT_ALL = False
if PLOT_ALL:
    n = spot_tuner.k
    for i in range(n-1):
        for j in range(i+1, n):
            spot_tuner.plot_contour(i=i, j=j, min_z=min_z, max_z = max_z)
```

21.10.6 Visualizing the Activation Distribution

i Reference:

- The following code is based on [\[PyTorch Lightning TUTORIAL 2: ACTIVATION FUNCTIONS\]](#), Author: Phillip Lippe, License: [\[CC BY-SA\]](#), Generated: 2023-03-15T09:52:39.179933.

After we have trained the models, we can look at the actual activation values that find inside the model. For instance, how many neurons are set to zero in ReLU? Where do we find most values in Tanh? To answer these questions, we can write a simple function which takes a trained model, applies it to a batch of images, and plots the histogram of the activations inside the network:

```
from spotPython.torch.activation import Sigmoid, Tanh, ReLU, LeakyReLU, ELU, Swish
act_fn_by_name = {"sigmoid": Sigmoid, "tanh": Tanh, "relu": ReLU, "leakyrelu": LeakyReLU,
```

```
from spotPython.hyperparameters.values import get_one_config_from_X
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
config = get_one_config_from_X(X, fun_control)
model = fun_control["core_model"](**config, _L_in=64, _L_out=11)
model
```

```
NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
```

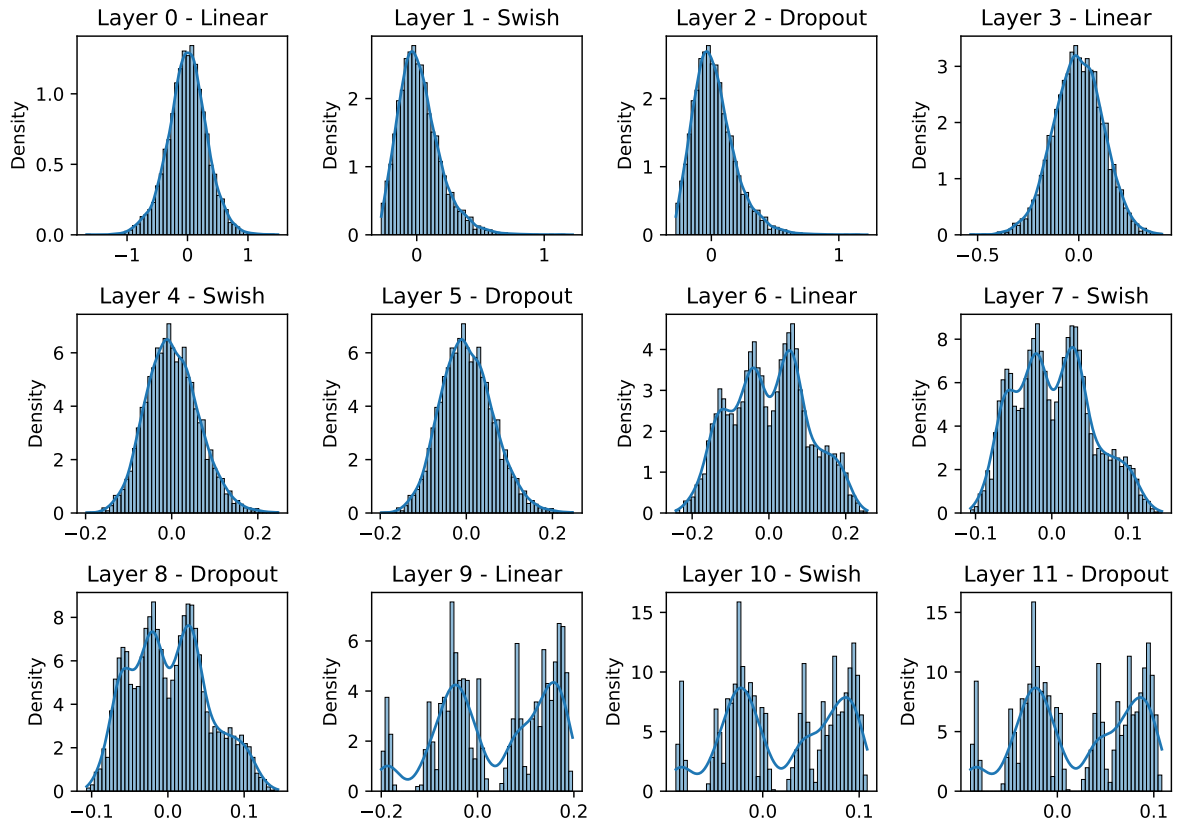
```

(1): Swish()
(2): Dropout(p=0.08985395193916346, inplace=False)
(3): Linear(in_features=64, out_features=32, bias=True)
(4): Swish()
(5): Dropout(p=0.08985395193916346, inplace=False)
(6): Linear(in_features=32, out_features=32, bias=True)
(7): Swish()
(8): Dropout(p=0.08985395193916346, inplace=False)
(9): Linear(in_features=32, out_features=16, bias=True)
(10): Swish()
(11): Dropout(p=0.08985395193916346, inplace=False)
(12): Linear(in_features=16, out_features=11, bias=True)
)
)

from spotPython.utils.eda import visualize_activations
visualize_activations(model, device="cpu", color=f"C{0}")

```


Activation distribution for activation function Swish()



22 Submission

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import OrdinalEncoder

import pandas as pd
from sklearn.preprocessing import OrdinalEncoder
train_df = pd.read_csv('./data/VBDP/train.csv', index_col=0)
# remove the id column
# train_df = train_df.drop(columns=['id'])
n_samples = train_df.shape[0]
n_features = train_df.shape[1] - 1
target_column = "prognosis"
# Encode our prognosis labels as integers for easier decoding later
enc = OrdinalEncoder()
y = enc.fit_transform(train_df[[target_column]])
test_df = pd.read_csv('./data/VBDP/test.csv', index_col=0)
test_df
```

id	sudden_fever	headache	mouth_bleed	nose_bleed	muscle_pain	joint_pain	vomiting	rash
707	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
708	1.0	1.0	0.0	1.0	0.0	1.0	1.0	1.0
709	1.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0
710	0.0	1.0	0.0	0.0	0.0	1.0	1.0	1.0
711	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0
712	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0
713	1.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0
714	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0
715	1.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0
716	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0
717	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0
718	1.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0
719	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0
720	1.0	0.0	0.0	1.0	0.0	1.0	1.0	1.0
721	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0
722	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
723	0.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0
724	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
725	1.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0
726	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0
727	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0
728	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0
729	1.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0
730	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
731	1.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0
732	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0
733	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0
734	0.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0
735	1.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0
736	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0
737	1.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0
738	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
739	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0
740	0.0	1.0	1.0	1.0	0.0	1.0	0.0	0.0
741	0.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0
742	1.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0
743	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
744	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0
745	1.0	1.0	1.0	0.0	1.0	0.0	0.0	1.0
746	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0
747	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0
748	0.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0
749	0.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0
750	0.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0
751	0.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0
752	0.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0
753	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0
754	1.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0
755	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0
756	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
757	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0

```

# convert the test_df to a torch tensor
X_tensor = torch.tensor(test_df.values, dtype=torch.float32)
X_tensor.shape

torch.Size([303, 64])

fun_control["device"]

'cpu'

y = model_loaded(X_tensor.to(fun_control["device"]))
y.shape

torch.Size([303, 11])

# convert the predictions to a numpy array
y = y.cpu().detach().numpy()
y

array([[1.95863433e-02, 4.95100431e-02, 2.55420476e-01, ...,
        1.21881925e-01, 1.13097653e-01, 3.91434878e-02],
       [9.96484041e-01, 4.40675512e-06, 2.70916062e-04, ...,
        6.82306127e-04, 4.47992898e-05, 6.18749527e-06],
       [2.92856384e-07, 1.98670546e-06, 9.08383057e-02, ...,
        9.08892632e-01, 1.78153823e-05, 4.64999317e-07],
       ...,
       [1.29832201e-09, 3.83390741e-08, 2.85736448e-03, ...,
        1.49473721e-07, 3.61995451e-04, 4.35826723e-06],
       [3.49802402e-04, 4.95580083e-04, 5.34949958e-01, ...,
        2.64983267e-01, 5.36504798e-02, 6.17286330e-03],
       [6.50187663e-04, 4.51166928e-03, 9.50201377e-02, ...,
        8.41977358e-01, 1.89952850e-02, 3.14282370e-03]], dtype=float32)

test_sorted_prediction_ids = np.argsort(-y, axis=1)
test_top_3_prediction_ids = test_sorted_prediction_ids[:, :3]
original_shape = test_top_3_prediction_ids.shape

```

```
test_top_3_prediction = enc.inverse_transform(test_top_3_prediction_ids.reshape(-1, 1))
test_top_3_prediction = test_top_3_prediction.reshape(original_shape)
test_df['prognosis'] = np.apply_along_axis(lambda x: np.array(' '.join(x), dtype="object"),
test_df['prognosis'].reset_index().to_csv('./data/VBDP/submission.csv', index=False)
```

23 Documentation of the Sequential Parameter Optimization

This document describes the `Spot` features.

23.1 Example: `spot`

```
import numpy as np
from math import inf
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from scipy.optimize import shgo
from scipy.optimize import direct
from scipy.optimize import differential_evolution
import matplotlib.pyplot as plt
```

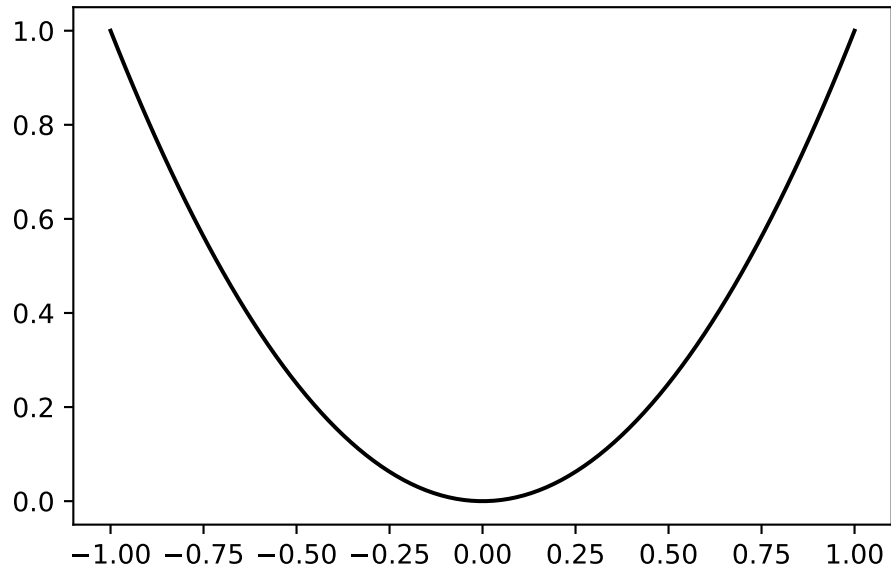
23.1.1 The Objective Function

The `spotPython` package provides several classes of objective functions. We will use an analytical objective function, i.e., a function that can be described by a (closed) formula:

$$f(x) = x^2$$

```
fun = analytical().fun_sphere

x = np.linspace(-1,1,100).reshape(-1,1)
y = fun(x)
plt.figure()
plt.plot(x,y, "k")
plt.show()
```



```
spot_1 = spot.Spot(fun=fun,
                    lower = np.array([-10]),
                    upper = np.array([100]),
                    fun_evals = 7,
                    fun_repeats = 1,
                    max_time = inf,
                    noise = False,
                    tolerance_x = np.sqrt(np.spacing(1)),
                    var_type=["num"],
                    infill_criterion = "y",
                    n_points = 1,
                    seed=123,
                    log_level = 50,
                    show_models=True,
                    fun_control = {},
                    design_control={"init_size": 5,
                                   "repeats": 1},
                    surrogate_control={"noise": False,
                                       "cod_type": "norm",
                                       "min_theta": -4,
                                       "max_theta": 3,
                                       "n_theta": 1,
                                       "model_optimizer": differential_evolution,
                                       "model_fun_evals": 1000,
```

})

spot's `__init__` method sets the control parameters. There are two parameter groups:

1. external parameters can be specified by the user
2. internal parameters, which are handled by `spot`.

23.1.2 External Parameters

external parameter	type	description	default	mandatory
<code>fun</code>	object	objective function		yes
<code>lower</code>	array	lower bound		yes
<code>upper</code>	array	upper bound		yes
<code>fun_evals</code>	int	number of function evaluations	15	no
<code>fun_evals</code>	int	number of function evaluations	15	no
<code>fun_control</code>	dict	noise etc.	{}	n
<code>max_time</code>	int	max run time budget	<code>inf</code>	no
<code>noise</code>	bool	if repeated evaluations of <code>fun</code> results in different values, then <code>noise</code> should be set to <code>True</code> .	<code>False</code>	no

external parameter	type	description	default	mandatory
<code>tolerance_x</code>	float	tolerance for new x solutions. Minimum distance of new solutions, generated by <code>suggest_new_X</code> , to already existing solutions. If zero (which is the default), every new solution is accepted.	0	no
<code>var_type</code>	list	list of type information, can be either "num" or "factor"	["num"]	no
<code>infill_criterion</code>	string	Can be "y", "s", "ei" (negative expected improvement), or "all"	"y"	no
<code>n_points</code>	int	number of infill points	1	no
<code>seed</code>	int	initial seed. If <code>Spot.run()</code> is called twice, different results will be generated. To reproduce results, the <code>seed</code> can be used.	123	no

external parameter	type	description	default	mandatory
log_level	int	log level with the following settings: NOTSET (0), DEBUG (10: Detailed information, typically of interest only when diagnosing problems.), INFO (20: Confirmation that things are working as expected.), WARNING (30: An indication that something unexpected happened, or indicative of some problem in the near future (e.g. 'disk space low'). The software is still working as expected.), ERROR (40: Due to a more serious problem, the software has not been able to perform some function.), and CRITICAL (50: A serious error, indicating that the program itself may be unable to continue running.)	50	no

external parameter	type	description	default	mandatory
<code>show_models</code>	bool	Plot model. Currently only 1-dim functions are supported	False	no
<code>design</code>	object	experimental design	None	no
<code>design_control</code>	dict	control parameters	see below	no
<code>surrogate</code>		surrogate model	kriging	no
<code>surrogate_control</code>	dict	control parameters	see below	no
<code>optimizer</code>	object	optimizer	see below	no
<code>optimizer_control</code>	dict	control parameters	see below	no

- Besides these single parameters, the following parameter dictionaries can be specified by the user:

- `fun_control`
- `design_control`
- `surrogate_control`
- `optimizer_control`

23.2 The `fun_control` Dictionary

external parameter	type	description	default	mandatory
<code>sigma</code>	float	noise: standard deviation	0	yes
<code>seed</code>	int	seed for rng	124	yes

23.3 The `design_control` Dictionary

external parameter	type	description	default	mandatory
<code>init_size</code>	int	initial sample size	10	yes

external parameter	type	description	default	mandatory
repeats	int	number of repeats of the initial sammples	1	yes

23.4 The surrogate_control Dictionary

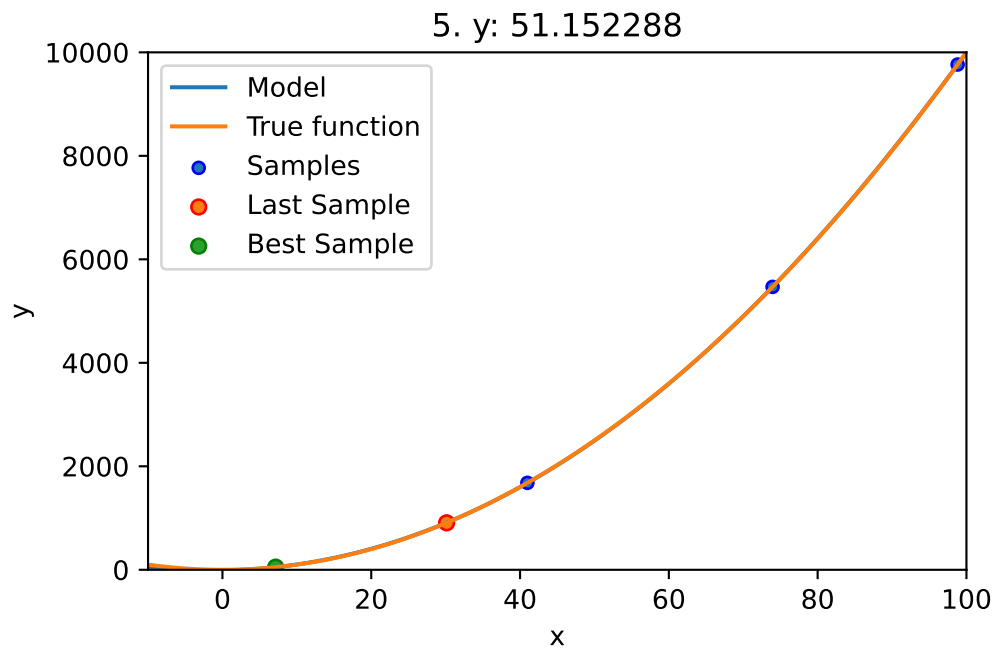
external parameter	type	description	default	mandatory
noise				
model_optimizer	object	optimizer	differential_evolution	
model_fun_evals				
min_theta			-3.	
max_theta			3.	
n_theta			1	
n_p			1	
optim_p			False	
cod_type			"norm"	
var_type				
use_cod_y	bool		False	

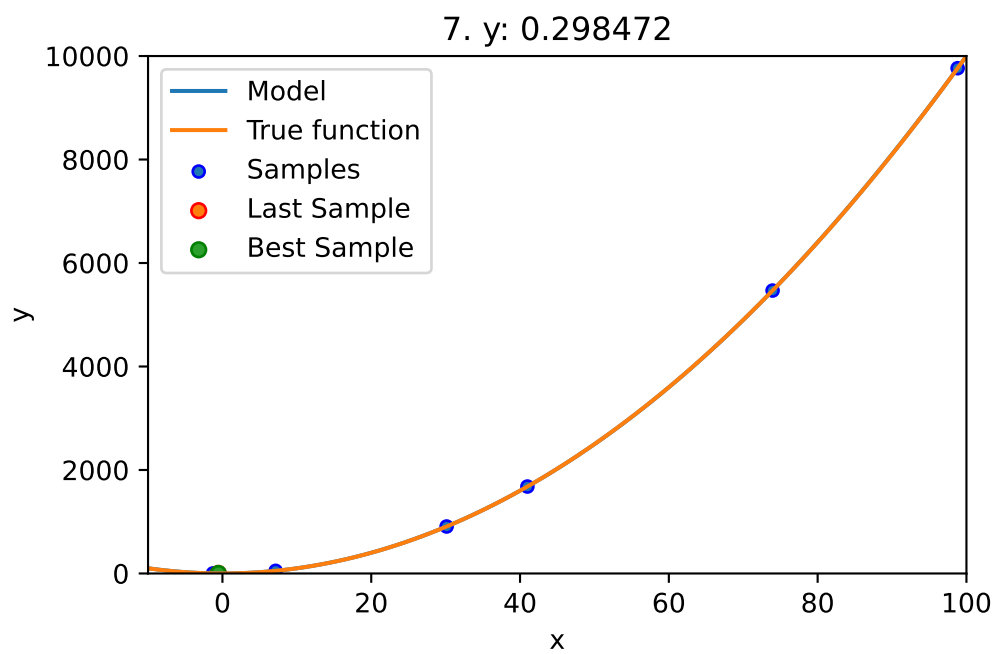
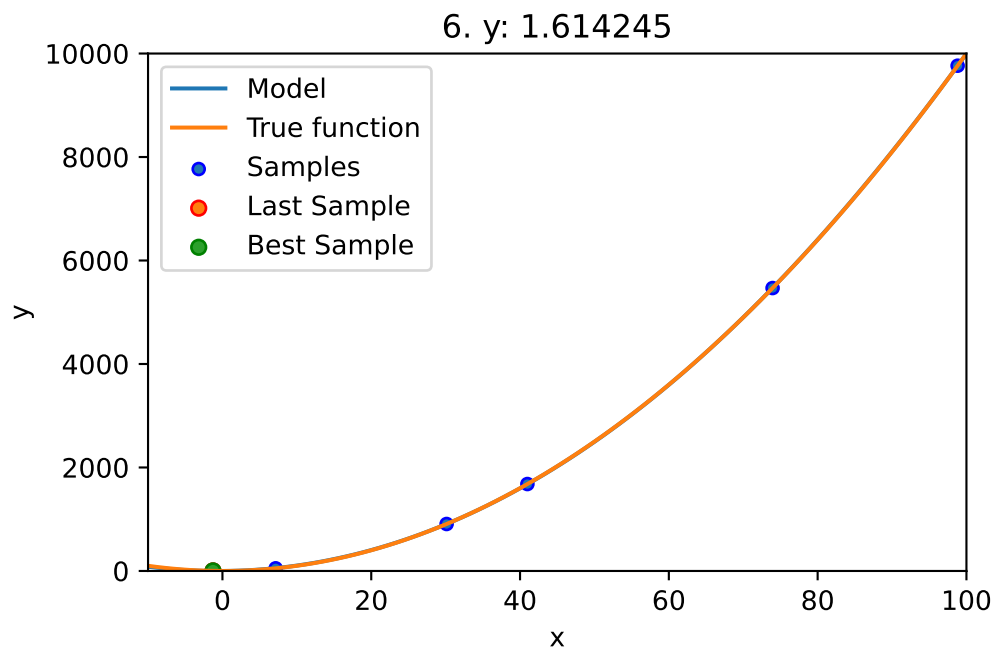
23.5 The optimizer_control Dictionary

external parameter	type	description	default	mandatory
max_iter	int	max number of iterations. Note: these are the cheap evaluations on the surrogate.	1000	no

23.6 Run

```
spot_1.run()
```





<spotPython.spot.spot.Spot at 0x158101360>

23.7 Print the Results

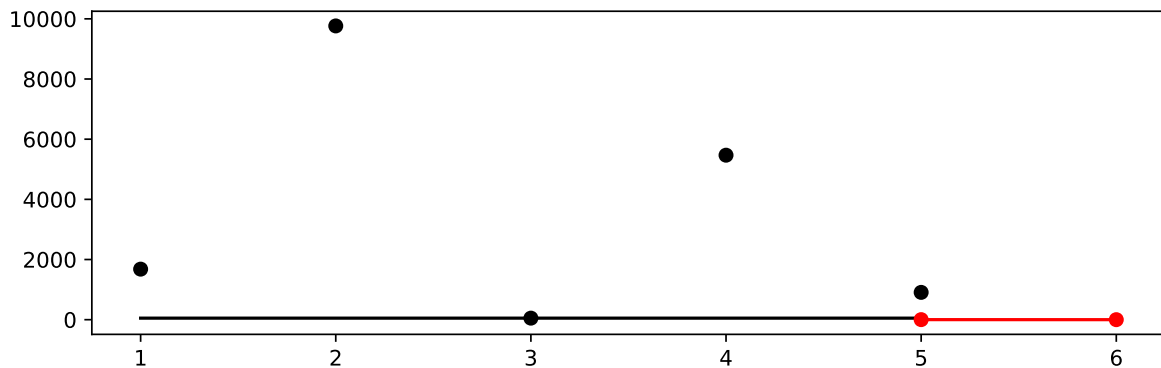
```
spot_1.print_results()
```

```
min y: 0.29847171516431314  
x0: -0.5463256493743572
```

```
[['x0', -0.5463256493743572]]
```

23.8 Show the Progress

```
spot_1.plot_progress()
```

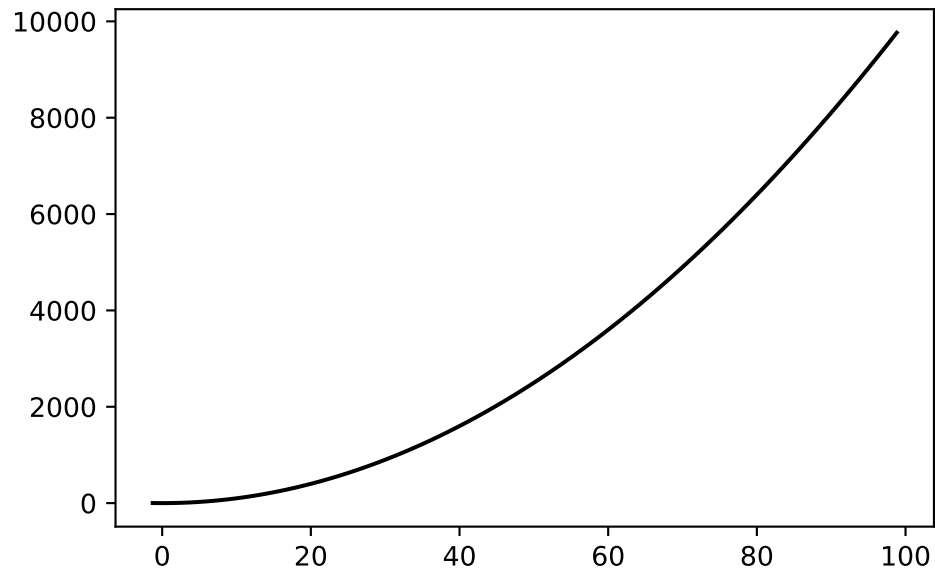


23.9 Visualize the Surrogate

- The plot method of the **kriging** surrogate is used.
- Note: the plot uses the interval defined by the ranges of the natural variables.

```
spot_1.surrogate.plot()
```

<Figure size 2700x1800 with 0 Axes>



23.10 Init: Build Initial Design

```
from spotPython.design.spacefilling import spacefilling
from spotPython.build.kriging import Kriging
from spotPython.fun.objectivefunctions import analytical
gen = spacefilling(2)
rng = np.random.RandomState(1)
lower = np.array([-5,-0])
upper = np.array([10,15])
fun = analytical().fun_branin
fun_control = {"sigma": 0,
               "seed": 123}

X = gen.scipy_lhd(10, lower=lower, upper = upper)
print(X)
y = fun(X, fun_control=fun_control)
print(y)
```

```
[[ 8.97647221 13.41926847]
 [ 0.66946019  1.22344228]
 [ 5.23614115 13.78185824]
 [ 5.6149825  11.5851384 ]
```



```

[-1.72963184  1.66516096]
[-4.26945568  7.1325531 ]
[ 1.26363761 10.17935555]
[ 2.88779942  8.05508969]
[-3.39111089  4.15213772]
[ 7.30131231  5.22275244]]
[128.95676449  31.73474356 172.89678121 126.71295908  64.34349975
 70.16178611  48.71407916  31.77322887  76.91788181  30.69410529]

```

23.11 Replicability

Seed

```

gen = spacefilling(2, seed=123)
X0 = gen.scipy_lhd(3)
gen = spacefilling(2, seed=345)
X1 = gen.scipy_lhd(3)
X2 = gen.scipy_lhd(3)
gen = spacefilling(2, seed=123)
X3 = gen.scipy_lhd(3)
X0, X1, X2, X3

```

```

(array([[0.77254938, 0.31539299],
        [0.59321338, 0.93854273],
        [0.27469803, 0.3959685 ]]),
 array([[0.78373509, 0.86811887],
        [0.06692621, 0.6058029 ],
        [0.41374778, 0.00525456]]),
 array([[0.121357  , 0.69043832],
        [0.41906219, 0.32838498],
        [0.86742658, 0.52910374]]),
 array([[0.77254938, 0.31539299],
        [0.59321338, 0.93854273],
        [0.27469803, 0.3959685 ]]))

```

23.12 Surrogates

23.12.1 A Simple Predictor

The code below shows how to use a simple model for prediction. Assume that only two (very costly) measurements are available:

1. $f(0) = 0.5$
2. $f(2) = 2.5$

We are interested in the value at $x_0 = 1$, i.e., $f(x_0 = 1)$, but cannot run an additional, third experiment.

```
from sklearn import linear_model
X = np.array([[0], [2]])
y = np.array([0.5, 2.5])
S_lm = linear_model.LinearRegression()
S_lm = S_lm.fit(X, y)
X0 = np.array([[1]])
y0 = S_lm.predict(X0)
print(y0)
```

[1.5]

Central Idea: Evaluation of the surrogate model S_{lm} is much cheaper (or / and much faster) than running the real-world experiment f .

23.13 Demo/Test: Objective Function Fails

SPOT expects `np.nan` values from failed objective function values. These are handled. Note: SPOT's counter considers only successful executions of the objective function.

```
import numpy as np
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
import numpy as np
from math import inf
# number of initial points:
ni = 20
# number of points
n = 30
```

```

fun = analytical().fun_random_error
lower = np.array([-1])
upper = np.array([1])
design_control={"init_size": ni}

spot_1 = spot.Spot(fun=fun,
                    lower = lower,
                    upper= upper,
                    fun_evals = n,
                    show_progress=False,
                    design_control=design_control,)
spot_1.run()
# To check whether the run was successfully completed,
# we compare the number of evaluated points to the specified
# number of points.
assert spot_1.y.shape[0] == n

[ 0.53176481 -0.9053821 -0.02203599 -0.21843718  0.78240941 -0.58120945
   nan  0.67234256  0.31802454          nan -0.75129705  0.97550354
   nan  0.0786237  0.82585329  0.23700598 -0.49274073 -0.82319082
 -0.17991251  0.1481835 ]
[-1.]

[-0.47259301]
[0.95541987]

[0.17335968]
[-0.58552368]

[-0.20126111]
[-0.60100809]

[-0.97897336]

[-0.2748985]

[0.8359486]
[0.99035591]

[0.01641232]
[0.5629346]

```

23.14 PyTorch: Detailed Description of the Data Splitting

23.14.1 Description of the "train_hold_out" Setting

The "train_hold_out" setting is used by default. It uses the loss function specified in `fun_control` and the metric specified in `fun_control`.

1. First, the method `HyperTorch().fun_torch` is called.
2. `fun_torc()`, which is implemented in the file `hypertorch.py`, calls `evaluate_hold_out()` as follows:

```
df_eval, _ = evaluate_hold_out(
    model,
    train_dataset=fun_control["train"],
    shuffle=self.fun_control["shuffle"],
    loss_function=self.fun_control["loss_function"],
    metric=self.fun_control["metric_torch"],
    device=self.fun_control["device"],
    show_batch_interval=self.fun_control["show_batch_interval"],
    path=self.fun_control["path"],
    task=self.fun_control["task"],
    writer=self.fun_control["writer"],
    writerId=config_id,
)
```

Note: Only the data set `fun_control["train"]` is used for training and validation. It is used in `evaluate_hold_out` as follows:

```
trainloader, valloader = create_train_val_data_loaders(
    dataset=train_dataset, batch_size=batch_size_instance, shuffle=shuffle
)
```

`create_train_val_data_loaders()` splits the `train_dataset` into `trainloader` and `valloader` using `torch.utils.data.random_split()` as follows:

```
def create_train_val_data_loaders(dataset, batch_size, shuffle, num_workers=0):
    test_abs = int(len(dataset) * 0.6)
    train_subset, val_subset = random_split(dataset, [test_abs, len(dataset) - test_abs])
    trainloader = torch.utils.data.DataLoader(
        train_subset, batch_size=int(batch_size), shuffle=shuffle, num_workers=num_workers
    )
    valloader = torch.utils.data.DataLoader(
```

```

        val_subset, batch_size=int(batch_size), shuffle=shuffle, num_workers=num_workers
    )
    return trainloader, valloader

```

The optimizer is set up as follows:

```

optimizer_instance = net.optimizer
lr_mult_instance = net.lr_mult
sgd_momentum_instance = net.sgd_momentum
optimizer = optimizer_handler(
    optimizer_name=optimizer_instance,
    params=net.parameters(),
    lr_mult=lr_mult_instance,
    sgd_momentum=sgd_momentum_instance,
)

```

3. `evaluate_hold_out()` sets the `net` attributes such as `epochs`, `batch_size`, `optimizer`, and `patience`. For each epoch, the methods `train_one_epoch()` and `validate_one_epoch()` are called, the former for training and the latter for validation and early stopping. The validation loss from the last epoch (not the best validation loss) is returned from `evaluate_hold_out`.
4. The method `train_one_epoch()` is implemented as follows:

```

def train_one_epoch(
    net,
    trainloader,
    batch_size,
    loss_function,
    optimizer,
    device,
    show_batch_interval=10_000,
    task=None,
):
    running_loss = 0.0
    epoch_steps = 0
    for batch_nr, data in enumerate(trainloader, 0):
        input, target = data
        input, target = input.to(device), target.to(device)
        optimizer.zero_grad()
        output = net(input)
        if task == "regression":

```

```

        target = target.unsqueeze(1)
        if target.shape == output.shape:
            loss = loss_function(output, target)
        else:
            raise ValueError(f"Shapes of target and output do not match:
                               {target.shape} vs {output.shape}")
    elif task == "classification":
        loss = loss_function(output, target)
    else:
        raise ValueError(f"Unknown task: {task}")
    loss.backward()
    torch.nn.utils.clip_grad_norm_(net.parameters(), max_norm=1.0)
    optimizer.step()
    running_loss += loss.item()
    epoch_steps += 1
    if batch_nr % show_batch_interval == (show_batch_interval - 1):
        print(
            "Batch: %5d. Batch Size: %d. Training Loss (running): %.3f"
            % (batch_nr + 1, int(batch_size), running_loss / epoch_steps)
        )
        running_loss = 0.0
    return loss.item()

```

5. The method `validate_one_epoch()` is implemented as follows:

```

def validate_one_epoch(net, valloader, loss_function, metric, device, task):
    val_loss = 0.0
    val_steps = 0
    total = 0
    correct = 0
    metric.reset()
    for i, data in enumerate(valloader, 0):
        # get batches
        with torch.no_grad():
            input, target = data
            input, target = input.to(device), target.to(device)
            output = net(input)
            # print(f"target: {target}")
            # print(f"output: {output}")
            if task == "regression":
                target = target.unsqueeze(1)

```

```

        if target.shape == output.shape:
            loss = loss_function(output, target)
        else:
            raise ValueError(f"Shapes of target and output
                             do not match: {target.shape} vs {output.shape}")
        metric_value = metric.update(output, target)
    elif task == "classification":
        loss = loss_function(output, target)
        metric_value = metric.update(output, target)
        _, predicted = torch.max(output.data, 1)
        total += target.size(0)
        correct += (predicted == target).sum().item()
    else:
        raise ValueError(f"Unknown task: {task}")
    val_loss += loss.cpu().numpy()
    val_steps += 1
loss = val_loss / val_steps
print(f"Loss on hold-out set: {loss}")
if task == "classification":
    accuracy = correct / total
    print(f"Accuracy on hold-out set: {accuracy}")
# metric on all batches using custom accumulation
metric_value = metric.compute()
metric_name = type(metric).__name__
print(f"{metric_name} value on hold-out data: {metric_value}")
return metric_value, loss

```

23.14.1.1 Description of the "test_hold_out" Setting

It uses the loss function specified in `fun_control` and the metric specified in `fun_control`.

1. First, the method `HyperTorch().fun_torch` is called.
2. `fun_torch()` calls `spotPython.torch.traintest.evaluate_hold_out()` similar to the "train_hold_out" setting with one exception: It passes an additional test data set to `evaluate_hold_out()` as follows:

```
test_dataset=fun_control["test"]
```

`evaluate_hold_out()` calls `create_train_test_data_loaders` instead of `create_train_val_data_loaders`: The two data sets are used in `create_train_test_data_loaders` as follows:

```

def create_train_test_data_loaders(dataset, batch_size, shuffle, test_dataset,
    num_workers=0):
    trainloader = torch.utils.data.DataLoader(
        dataset, batch_size=int(batch_size), shuffle=shuffle,
        num_workers=num_workers
    )
    testloader = torch.utils.data.DataLoader(
        test_dataset, batch_size=int(batch_size), shuffle=shuffle,
        num_workers=num_workers
    )
    return trainloader, testloader

```

3. The following steps are identical to the "train_hold_out" setting. Only a different data loader is used for testing.

23.14.1.2 Detailed Description of the "train_cv" Setting

It uses the loss function specified in `fun_control` and the metric specified in `fun_control`.

1. First, the method `HyperTorch().fun_torch` is called.
2. `fun_torch()` calls `spotPython.torch.traintest.evaluate_cv()` as follows (Note: Only the data set `fun_control["train"]` is used for CV.):

```

df_eval, _ = evaluate_cv(
    model,
    dataset=fun_control["train"],
    shuffle=self.fun_control["shuffle"],
    device=self.fun_control["device"],
    show_batch_interval=self.fun_control["show_batch_interval"],
    task=self.fun_control["task"],
    writer=self.fun_control["writer"],
    writerId=config_id,
)

```

3. In `evaluate_cv()`, the following steps are performed: The optimizer is set up as follows:

```

optimizer_instance = net.optimizer
lr_instance = net.lr
sgd_momentum_instance = net.sgd_momentum
optimizer = optimizer_handler(optimizer_name=optimizer_instance,
    params=net.parameters(), lr_mult=lr_mult_instance)

```


`evaluate_cv()` sets the `net` attributes such as `epochs`, `batch_size`, `optimizer`, and `patience`. CV is implemented as follows:

```
def evaluate_cv(
    net,
    dataset,
    shuffle=False,
    loss_function=None,
    num_workers=0,
    device=None,
    show_batch_interval=10_000,
    metric=None,
    path=None,
    task=None,
    writer=None,
    writerId=None,
):
    lr_mult_instance = net.lr_mult
    epochs_instance = net.epochs
    batch_size_instance = net.batch_size
    k_folds_instance = net.k_folds
    optimizer_instance = net.optimizer
    patience_instance = net.patience
    sgd_momentum_instance = net.sgd_momentum
    removed_attributes, net = get_removed_attributes_and_base_net(net)
    metric_values = {}
    loss_values = {}
    try:
        device = getDevice(device=device)
        if torch.cuda.is_available():
            device = "cuda:0"
            if torch.cuda.device_count() > 1:
                print("We will use", torch.cuda.device_count(), "GPUs!")
                net = nn.DataParallel(net)
        net.to(device)
        optimizer = optimizer_handler(
            optimizer_name=optimizer_instance,
            params=net.parameters(),
            lr_mult=lr_mult_instance,
            sgd_momentum=sgd_momentum_instance,
        )
        kfold = KFold(n_splits=k_folds_instance, shuffle=shuffle)
```

```

for fold, (train_ids, val_ids) in enumerate(kfold.split(dataset)):
    print(f"Fold: {fold + 1}")
    train_subsampler = torch.utils.data.SubsetRandomSampler(train_ids)
    val_subsampler = torch.utils.data.SubsetRandomSampler(val_ids)
    trainloader = torch.utils.data.DataLoader(
        dataset, batch_size=batch_size_instance,
        sampler=train_subsampler, num_workers=num_workers
    )
    valloader = torch.utils.data.DataLoader(
        dataset, batch_size=batch_size_instance,
        sampler=val_subsampler, num_workers=num_workers
    )
    # each fold starts with new weights:
    reset_weights(net)
    # Early stopping parameters
    best_val_loss = float("inf")
    counter = 0
    for epoch in range(epochs_instance):
        print(f"Epoch: {epoch + 1}")
        # training loss from one epoch:
        training_loss = train_one_epoch(
            net=net,
            trainloader=trainloader,
            batch_size=batch_size_instance,
            loss_function=loss_function,
            optimizer=optimizer,
            device=device,
            show_batch_interval=show_batch_interval,
            task=task,
        )
        # Early stopping check. Calculate validation loss from one epoch:
        metric_values[fold], loss_values[fold] = validate_one_epoch(
            net, valloader=valloader, loss_function=loss_function,
            metric=metric, device=device, task=task
        )
        # Log the running loss averaged per batch
        metric_name = "Metric"
        if metric is None:
            metric_name = type(metric).__name__
            print(f"{metric_name} value on hold-out data:
                  {metric_values[fold]}")

```

```

        if writer is not None:
            writer.add_scalars(
                "evaluate_cv fold:" + str(fold + 1) +
                ". Train & Val Loss and Val Metric" + writerId,
                {"Train loss": training_loss, "Val loss":
                 loss_values[fold], metric_name: metric_values[fold]},
                epoch + 1,
            )
            writer.flush()
        if loss_values[fold] < best_val_loss:
            best_val_loss = loss_values[fold]
            counter = 0
            # save model:
            if path is not None:
                torch.save(net.state_dict(), path)
        else:
            counter += 1
            if counter >= patience_instance:
                print(f"Early stopping at epoch {epoch}")
                break

    df_eval = sum(loss_values.values()) / len(loss_values.values())
    df_metrics = sum(metric_values.values()) / len(metric_values.values())
    df_preds = np.nan
except Exception as err:
    print(f"Error in Net_Core. Call to evaluate_cv() failed. {err=},
          {type(err)=}")
    df_eval = np.nan
    df_preds = np.nan
add_attributes(net, removed_attributes)
if writer is not None:
    metric_name = "Metric"
    if metric is None:
        metric_name = type(metric).__name__
    writer.add_scalars(
        "CV: Val Loss and Val Metric" + writerId,
        {"CV-loss": df_eval, metric_name: df_metrics},
        epoch + 1,
    )
    writer.flush()
return df_eval, df_preds, df_metrics

```

4. The method `train_fold()` is implemented as shown above.

5. The method `validate_one_epoch()` is implemented as shown above. In contrast to the hold-out setting, it is called for each of the k folds. The results are stored in a dictionaries `metric_values` and `loss_values`. The results are averaged over the k folds and returned as `df_eval`.

23.14.1.3 Detailed Description of the "test_cv" Setting

It uses the loss function specified in `fun_control` and the metric specified in `fun_control`.

1. First, the method `HyperTorch().fun_torch` is called.
2. `fun_torch()` calls `spotPython.torch.traintest.evaluate_cv()` as follows:

```
df_eval, _ = evaluate_cv(  
    model,  
    dataset=fun_control["test"],  
    shuffle=self.fun_control["shuffle"],  
    device=self.fun_control["device"],  
    show_batch_interval=self.fun_control["show_batch_interval"],  
    task=self.fun_control["task"],  
    writer=self.fun_control["writer"],  
    writerId=config_id,  
)
```

Note: The data set `fun_control["test"]` is used for CV. The rest is the same as for the "train_cv" setting.

23.14.1.4 Detailed Description of the Final Model Training and Evaluation

There are two methods that can be used for the final evaluation of a Pytorch model:

1. "train_tuned and
2. "test_tuned".

`train_tuned()` is just a wrapper to `evaluate_hold_out` using the `train` data set. It is implemented as follows:

```
def train_tuned(  
    net,  
    train_dataset,  
    shuffle,  
    loss_function,  
    metric,
```

```

        device=None,
        show_batch_interval=10_000,
        path=None,
        task=None,
        writer=None,
    ):
        evaluate_hold_out(
            net=net,
            train_dataset=train_dataset,
            shuffle=shuffle,
            test_dataset=None,
            loss_function=loss_function,
            metric=metric,
            device=device,
            show_batch_interval=show_batch_interval,
            path=path,
            task=task,
            writer=writer,
        )

```

The `test_tuned()` procedure is implemented as follows:

```

def test_tuned(net, shuffle, test_dataset=None, loss_function=None,
               metric=None, device=None, path=None, task=None):
    batch_size_instance = net.batch_size
    removed_attributes, net = get_removed_attributes_and_base_net(net)
    if path is not None:
        net.load_state_dict(torch.load(path))
        net.eval()
    try:
        device = getDevice(device=device)
        if torch.cuda.is_available():
            device = "cuda:0"
            if torch.cuda.device_count() > 1:
                print("We will use", torch.cuda.device_count(), "GPUs!")
                net = nn.DataParallel(net)
        net.to(device)
        valloader = torch.utils.data.DataLoader(
            test_dataset, batch_size=int(batch_size_instance),
            shuffle=shuffle,
            num_workers=0
        )
    )

```

```

metric_value, loss = validate_one_epoch(
    net, valloader=valloader, loss_function=loss_function,
    metric=metric, device=device, task=task
)
df_eval = loss
df_metric = metric_value
df_preds = np.nan
except Exception as err:
    print(f"Error in Net_Core. Call to test_tuned() failed. {err=},
          {type(err)=}")
    df_eval = np.nan
    df_metric = np.nan
    df_preds = np.nan
add_attributes(net, removed_attributes)
print(f"Final evaluation: Validation loss: {df_eval}")
print(f"Final evaluation: Validation metric: {df_metric}")
print("-----")
return df_eval, df_preds, df_metric

```

References

- Bartz, Eva, Thomas Bartz-Beielstein, Martin Zaefferer, and Olaf Mersmann, eds. 2022. *Hyperparameter Tuning for Machine and Deep Learning with R - A Practical Guide*. Springer.
- Bartz-Beielstein, Thomas. 2023. “PyTorch Hyperparameter Tuning with SPOT: Comparison with Ray Tuner and Default Hyperparameters on CIFAR10.” https://github.com/sequential-parameter-optimization/spotPython/blob/main/notebooks/14_spot_ray_hpt_torch_cifar10.ipynb.
- Bartz-Beielstein, Thomas, Jürgen Branke, Jörn Mehnen, and Olaf Mersmann. 2014. “Evolutionary Algorithms.” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 4 (3): 178–95.
- Bartz-Beielstein, Thomas, Carola Doerr, Jakob Bossek, Sowmya Chandrasekaran, Tome Eftimov, Andreas Fischbach, Pascal Kerschke, et al. 2020. “Benchmarking in Optimization: Best Practice and Open Issues.” arXiv. <https://arxiv.org/abs/2007.03488>.
- Bartz-Beielstein, Thomas, Christian Lasarczyk, and Mike Preuss. 2005. “Sequential Parameter Optimization.” In *Proceedings 2005 Congress on Evolutionary Computation (CEC’05), Edinburgh, Scotland*, edited by B McKay et al., 773–80. Piscataway NJ: IEEE Press.
- Lewis, R M, V Torczon, and M W Trosset. 2000. “Direct search methods: Then and now.” *Journal of Computational and Applied Mathematics* 124 (1–2): 191–207.
- Li, Lisha, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2016. “Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization.” *arXiv e-Prints*, March, arXiv:1603.06560.
- Meignan, David, Sigrid Knust, Jean-Marc Frayet, Gilles Pesant, and Nicolas Gaud. 2015. “A Review and Taxonomy of Interactive Optimization Methods in Operations Research.” *ACM Transactions on Interactive Intelligent Systems*, September.
- Montiel, Jacob, Max Halford, Saulo Martiello Mastelini, Geoffrey Bolmier, Raphael Sourty, Robin Vaysse, Adil Zouitine, et al. 2021. “River: Machine Learning for Streaming Data in Python.”
- PyTorch. 2023a. “Hyperparameter Tuning with Ray Tune.” https://pytorch.org/tutorials/beginner/hyperparameter_tuning_tutorial.html.
- . 2023b. “Training a Classifier.” https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html.