

Hyperparameter Tuning Cookbook

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

Thomas Bartz-Beielstein

Jun 28, 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	138
10.9.7	Plot all Combinations of Hyperparameters	138
11	HPT: PyTorch With fashionMNIST	139
11.1	Step 1: Setup	139
11.2	Step 2: Initialization of the Empty <code>fun_control</code> Dictionary	141
11.3	Step 3: PyTorch Data Loading	141
11.3.1	Load fashionMNIST Data	141
11.4	Step 4: Specification of the Preprocessing Model	142

11.5	Step 5: Select Model (<code>algorithm</code>) and <code>core_model_hyper_dict</code>	142
11.5.1	The Search Space	143
11.5.2	Configuring the Search Space With <code>spotPython</code>	143
11.6	Step 6: Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	145
11.6.1	Modify hyperparameter of type numeric and integer (boolean)	145
11.6.2	Modify hyperparameter of type factor	146
11.6.3	Optimizers	146
11.7	Step 7: Selection of the Objective (Loss) Function	146
11.7.1	Evaluation	146
11.7.2	Metric	147
11.8	Step 8: Calling the SPOT Function	147
11.8.1	Preparing the SPOT Call	147
11.8.2	The Objective Function <code>fun_torch</code>	148
11.8.3	Starting the Hyperparameter Tuning	148
11.9	Step 9: Tensorboard	153
11.10	Step 10: Results	153
11.10.1	Show variable importance	154
11.10.2	Get the Tuned Architecture (SPOT Results)	155
11.10.3	Get Default Hyperparameters	156
11.10.4	Evaluation of the Default and the Tuned Architectures	156
11.10.5	Detailed Hyperparameter Plots	159
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	181
12.10.7	Plot all Combinations of Hyperparameters	181
13	HPT: River	182
13.1	Step 1: Setup	182
13.1.1	<code>river</code> Hyperparameter Tuning: HATR with Friedman Drift Data . . .	182
13.2	Step 2: Initialization of the <code>fun_control</code> Dictionary	183
13.3	Step 3: Load the Friedman Drift Data	183
13.4	Step 4: Specification of the Preprocessing Model	184
13.5	Step 5: Select <code>algorithm</code> and <code>core_model_hyper_dict</code>	185
13.6	Step 6: Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	188
13.6.1	Modify hyperparameter of type factor	188
13.6.2	Modify hyperparameter of type numeric and integer (boolean)	188
13.7	Step 7: Selection of the Objective (Loss) Function	188
13.8	Step 8: Calling the SPOT Function	189
13.8.1	Prepare the SPOT Parameters	189
13.8.2	Run the <code>Spot</code> Optimizer	190
13.9	Step 9: Results	191
13.9.1	Show variable importance	193
13.9.2	Build and Evaluate HTR Model with Tuned Hyperparameters	193
13.9.3	The Large Data Set (k=0.2)	194
13.9.4	Get Default Hyperparameters	195
13.9.5	Get SPOT Results	198
13.9.6	Visualize Regression Trees	202
13.9.7	Spot Model	202
13.9.8	Detailed Hyperparameter Plots	204
13.9.9	Parallel Coordinates Plots	205
13.9.10	Plot all Combinations of Hyperparameters	205

14 HPT: PyTorch With spotPython and Ray Tune on CIFAR10	206
14.1 Step 1: Setup	207
14.2 Step 2: Initialization of the <code>fun_control</code> Dictionary	208
14.3 Step 3: PyTorch Data Loading	208
14.4 Step 4: Specification of the Preprocessing Model	209
14.5 Step 5: Select Model (<code>algorithm</code>) and <code>core_model_hyper_dict</code>	210
14.5.1 The <code>Net_Core</code> class	212
14.5.2 Comparison of the Approach Described in the PyTorch Tutorial With <code>spotPython</code>	212
14.5.3 The Search Space: Hyperparameters	213
14.5.4 Configuring the Search Space With Ray Tune	213
14.5.5 Configuring the Search Space With <code>spotPython</code>	214
14.6 Step 6: Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	216
14.6.1 Optimizers	217
14.7 Step 7: Selection of the Objective (Loss) Function	219
14.7.1 Evaluation: Data Splitting	219
14.7.2 Hold-out Data Split	219
14.7.3 Cross-Validation	220
14.7.4 Overview of the Evaluation Settings	220
14.7.5 Evaluation: Loss Functions and Metrics	222
14.8 Step 8: Calling the SPOT Function	223
14.8.1 Preparing the SPOT Call	223
14.8.2 The Objective Function <code>fun_torch</code>	224
14.8.3 Using Default Hyperparameters or Results from Previous Runs	224
14.8.4 Starting the Hyperparameter Tuning	224
14.9 Step 9: Tensorboard	233
14.9.1 Tensorboard: Start Tensorboard	233
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	245
14.12 Appendix	245
14.12.1 Sample Output From Ray Tune's Run	245
15 HPT: sklearn RandomForestClassifier VBDP Data	247
15.1 Step 1: Setup	247
15.2 Step 2: Initialization of the Empty <code>fun_control</code> Dictionary	248

15.3	Step 3: PyTorch Data Loading	249
15.3.1	Load Data: Classification VBDP	249
15.3.2	Holdout Train and Test Data	249
15.4	Step 4: Specification of the Preprocessing Model	250
15.5	Step 5: Select Model (<code>algorithm</code>) and <code>core_model_hyper_dict</code>	251
15.6	Step 6: Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	253
15.6.1	Modify hyperparameter of type numeric and integer (boolean)	253
15.6.2	Modify hyperparameter of type factor	253
15.6.3	Optimizers	254
15.6.4	Selection of the Objective: Metric and Loss Functions	254
15.7	Step 7: Selection of the Objective (Loss) Function	254
15.7.1	Metric Function	254
15.7.2	Evaluation on Hold-out Data	255
15.7.3	OOB Score	256
15.8	Step 8: Calling the SPOT Function	257
15.8.1	Preparing the SPOT Call	257
15.8.2	The Objective Function	257
15.8.3	Run the <code>Spot</code> Optimizer	258
15.9	Step 9: Tensorboard	260
15.10	Step 10: Results	260
15.10.1	Show variable importance	261
15.10.2	Get Default Hyperparameters	262
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	265
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	274
15.10.11	Plot all Combinations of Hyperparameters	274
16	HPT: sklearn XGB Classifier VBDP Data	275
16.1	Step 1: Setup	275
16.2	Step 2: Initialization of the Empty <code>fun_control</code> Dictionary	276
16.3	Step 3: PyTorch Data Loading	277
16.3.1	1. Load Data: Classification VBDP	277
16.3.2	Holdout Train and Test Data	277
16.4	Step 4: Specification of the Preprocessing Model	278
16.5	Step 5: Select Model (<code>algorithm</code>) and <code>core_model_hyper_dict</code>	279

16.6	Step 6: Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	281
16.6.1	Modify hyperparameter of type numeric and integer (boolean)	281
16.6.2	Modify hyperparameter of type factor	281
16.6.3	Optimizers	282
16.7	Step 7: Selection of the Objective (Loss) Function	282
16.7.1	Evaluation	282
16.7.2	Selection of the Objective: Metric and Loss Functions	282
16.7.3	Loss Function	282
16.7.4	Metric Function	282
16.7.5	Evaluation on Hold-out Data	284
16.8	Step 8: Calling the SPOT Function	284
16.8.1	Preparing the SPOT Call	284
16.8.2	The Objective Function	285
16.8.3	Run the <code>Spot</code> Optimizer	285
16.9	Step 9: Tensorboard	287
16.10	Step 10: Results	288
16.10.1	Show variable importance	289
16.10.2	Get Default Hyperparameters	289
16.10.3	Get SPOT Results	290
16.10.4	Evaluate SPOT Results	291
16.10.5	Handling Non-deterministic Results	292
16.10.6	Evaluation of the Default Hyperparameters	292
16.10.7	Plot: Compare Predictions	293
16.10.8	Cross-validated Evaluations	295
16.10.9	Detailed Hyperparameter Plots	296
16.10.10	Parallel Coordinates Plot	299
16.10.11	Plot all Combinations of Hyperparameters	299
17	HPT: sklearn SVC VBDP Data	300
17.1	Step 1: Setup	300
17.2	Step 2: Initialization of the Empty <code>fun_control</code> Dictionary	301
17.3	Step 3: PyTorch Data Loading	302
17.3.1	1. Load Data: Classification VBDP	302
17.3.2	Holdout Train and Test Data	302
17.4	Step 4: Specification of the Preprocessing Model	303
17.5	Step 5: Select Model (<code>algorithm</code>) and <code>core_model_hyper_dict</code>	304
17.6	Step 6: Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	306
17.6.1	Modify hyperparameter of type numeric and integer (boolean)	306
17.6.2	Modify hyperparameter of type factor	306
17.6.3	Optimizers	306
17.6.4	Selection of the Objective: Metric and Loss Functions	307

17.7	Step 7: Selection of the Objective (Loss) Function	307
17.7.1	Metric Function	307
17.7.2	Evaluation on Hold-out Data	308
17.8	Step 8: Calling the SPOT Function	309
17.8.1	Preparing the SPOT Call	309
17.8.2	The Objective Function	310
17.8.3	Run the <code>Spot</code> Optimizer	310
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	326
17.10.11	Plot all Combinations of Hyperparameters	326
18	HPT: sklearn KNN Classifier VBDP Data	327
18.1	Step 1: Setup	327
18.2	Step 2: Initialization of the Empty <code>fun_control</code> Dictionary	328
18.2.1	Load Data: Classification VBDP	328
18.2.2	Holdout Train and Test Data	329
18.3	Step 4: Specification of the Preprocessing Model	330
18.4	Step 5: Select Model (<code>algorithm</code>) and <code>core_model_hyper_dict</code>	331
18.5	Step 6: Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	332
18.5.1	Modify hyperparameter of type numeric and integer (boolean)	332
18.5.2	Modify hyperparameter of type factor	333
18.5.3	Optimizers	333
18.5.4	Selection of the Objective: Metric and Loss Functions	333
18.6	Step 7: Selection of the Objective (Loss) Function	333
18.6.1	Metric Function	334
18.6.2	Evaluation on Hold-out Data	335
18.7	Step 8: Calling the SPOT Function	335
18.7.1	Preparing the SPOT Call	335
18.7.2	The Objective Function	336
18.7.3	Run the <code>Spot</code> Optimizer	336
18.8	Step 9: Tensorboard	340

18.9	Step 10: Results	340
18.9.1	Show variable importance	341
18.9.2	Get Default Hyperparameters	342
18.9.3	Get SPOT Results	343
18.9.4	Evaluate SPOT Results	343
18.9.5	Handling Non-deterministic Results	344
18.9.6	Evaluation of the Default Hyperparameters	345
18.9.7	Plot: Compare Predictions	345
18.9.8	Cross-validated Evaluations	347
18.9.9	Detailed Hyperparameter Plots	348
18.9.10	Parallel Coordinates Plot	351
18.9.11	Plot all Combinations of Hyperparameters	351
19	HPT PyTorch: Regression	352
19.1	Step 1: Setup	352
19.2	Step 2: Initialization of the <code>fun_control</code> Dictionary	354
19.3	Step 3: PyTorch Data Loading	354
19.4	Step 4: Specification of the Preprocessing Model	356
19.5	Step 5: Select Model (<code>algorithm</code>) and <code>core_model_hyper_dict</code>	356
19.5.1	Implementing a Configurable Neural Network With <code>spotPython</code>	356
19.5.2	The Search Space	358
19.5.3	Configuring the Search Space With <code>spotPython</code>	358
19.6	Step 6: Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	360
19.6.1	Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	360
19.6.2	Optimizers	360
19.7	Step 7: Selection of the Objective (Loss) Function	361
19.7.1	Evaluation	361
19.7.2	Loss Functions and Metrics	361
19.7.3	Metric	361
19.8	Step 8: Calling the SPOT Function	361
19.8.1	Preparing the SPOT Call	361
19.8.2	The Objective Function <code>fun_torch</code>	362
19.8.3	Starting the Hyperparameter Tuning	362
19.9	Step 9: Tensorboard	455
19.10	Step 10: Results	455
19.10.1	Get the Tuned Architecture (SPOT Results)	456
19.10.2	Evaluation of the Tuned Architecture	457
19.10.3	Cross-validated Evaluations	461
19.10.4	Detailed Hyperparameter Plots	495
19.10.5	Parallel Coordinates Plot	497
19.11	Summary and Outlook	497

20 HPT: PyTorch With VBDP	499
20.1 Step 1: Setup	500
20.2 Step 2: Initialization of the <code>fun_control</code> Dictionary	501
20.3 Step 3: PyTorch Data Loading	501
20.3.1 1. Load VBDP Data	501
20.3.2 Check content of the target column	502
20.4 Step 4: Specification of the Preprocessing Model	503
20.5 Step 5: Select <code>algorithm</code> and <code>core_model_hyper_dict</code>	504
20.5.1 Implementing a Configurable Neural Network With <code>spotPython</code>	504
20.5.2 Add the NN Model to the <code>fun_control</code> Dictionary	504
20.6 Step 6: Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	506
20.6.1 Optimizers	506
20.7 Step 7: Selection of the Objective (Loss) Function	507
20.7.1 Evaluation	507
20.7.2 Loss Functions and Metrics	507
20.7.3 Metric	507
20.8 Step 8: Calling the SPOT Function	508
20.8.1 Preparing the SPOT Call	508
20.8.2 The Objective Function <code>fun_torch</code>	509
20.8.3 Starting the Hyperparameter Tuning	509
20.9 Step 9: Tensorboard	515
20.10 Step 10: Results	515
20.10.1 Get the Tuned Architecture	517
20.10.2 Evaluation of the Tuned Architecture	518
20.10.3 Cross-validated Evaluations	519
20.10.4 Detailed Hyperparameter Plots	522
20.10.5 Parallel Coordinates Plot	524
20.10.6 Plot all Combinations of Hyperparameters	524
 21 HPT PyTorch Lightning: VBDP	 525
21.1 Step 1: Setup	526
21.2 Step 2: Initialization of the <code>fun_control</code> Dictionary	527
21.3 Step 3: PyTorch Data Loading	527
21.3.1 Lightning Dataset and <code>DataModule</code>	527
21.4 Step 4: Specification of the Preprocessing Model	529
21.5 Step 5: Select the NN Model (<code>algorithm</code>) and <code>core_model_hyper_dict</code>	529
21.5.1 Implementing a Configurable Neural Network With <code>spotPython</code>	529
21.5.2 Add the NN Model to the <code>fun_control</code> Dictionary	530
21.6 Step 6: Modify <code>hyper_dict</code> Hyperparameters for the Selected Algorithm aka <code>core_model</code>	532
21.7 Step 7: Data Splitting, the Objective (Loss) Function and the Metric	534
21.7.1 Evaluation	534

21.7.2	Loss Functions and Metrics	534
21.7.3	Metric	534
21.8	Step 8: Calling the SPOT Function	535
21.8.1	Preparing the SPOT Call	535
21.8.2	The Objective Function <code>fun</code>	536
21.8.3	Starting the Hyperparameter Tuning	536
21.9	Step 9: Tensorboard	554
21.10	Step 10: Results	554
21.10.1	Get the Tuned Architecture	556
21.10.2	Cross Validation With Lightning	557
21.10.3	Detailed Hyperparameter Plots	566
21.10.4	Parallel Coordinates Plot	570
21.10.5	Plot all Combinations of Hyperparameters	571
21.10.6	Visualizing the Activation Distribution	571
22	Documentation of the Sequential Parameter Optimization	574
22.1	Example: <code>spot</code>	574
22.1.1	The Objective Function	574
22.1.2	External Parameters	576
22.2	The <code>fun_control</code> Dictionary	579
22.3	The <code>design_control</code> Dictionary	579
22.4	The <code>surrogate_control</code> Dictionary	580
22.5	The <code>optimizer_control</code> Dictionary	580
22.6	Run	581
22.7	Print the Results	583
22.8	Show the Progress	583
22.9	Visualize the Surrogate	583
22.10	Init: Build Initial Design	584
22.11	Replicability	585
22.12	Surrogates	586
22.12.1	A Simple Predictor	586
22.13	Demo/Test: Objective Function Fails	586
22.14	PyTorch: Detailed Description of the Data Splitting	589
22.14.1	Description of the " <code>train_hold_out</code> " Setting	589
References		600

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.

Chapter 21 shows the integration of `spotPython` into the PyTorch Lightning training workflow.

Section 14.11 presents a summary and an outlook.

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

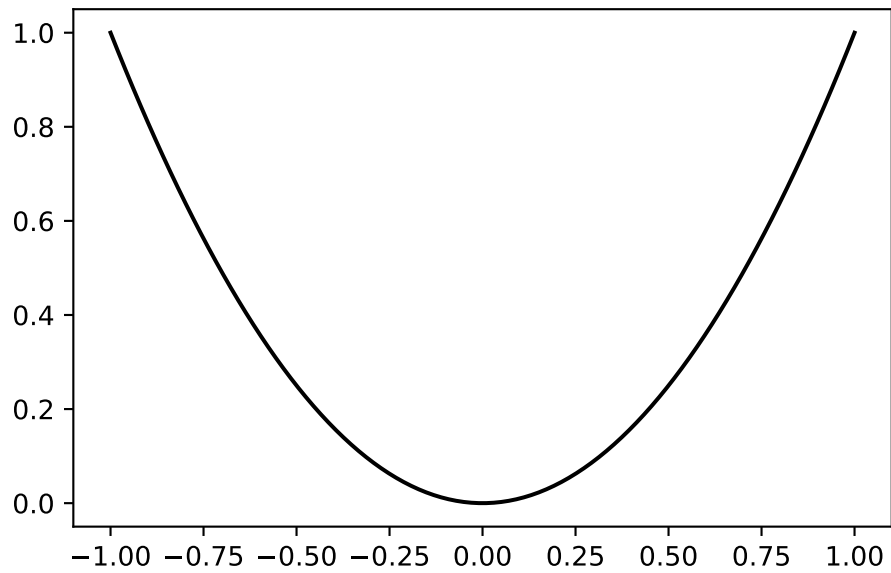
The `spotPython` package provides several classes of objective functions. We will use an analytical objective function, i.e., a function that can be described by a (closed) formula:

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

```
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()
```

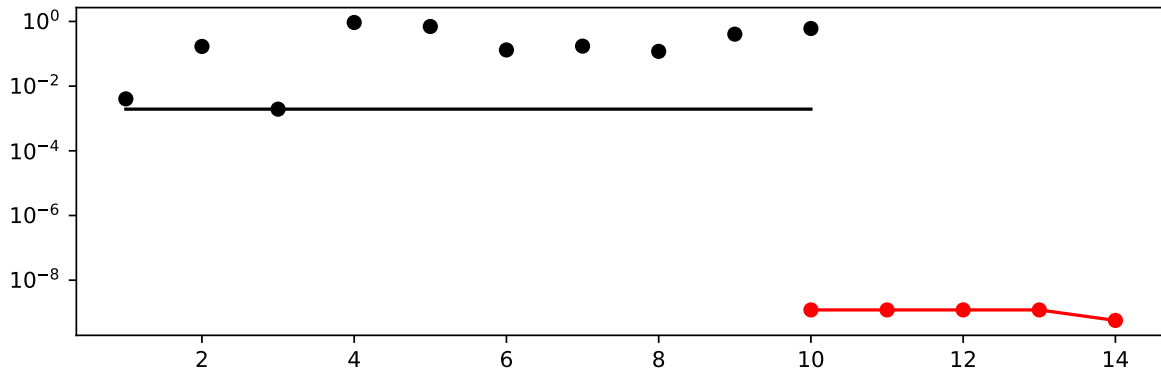
```
<spotPython.spot.spot.Spot at 0x15ed02590>
```

```
spot_0.print_results()
```

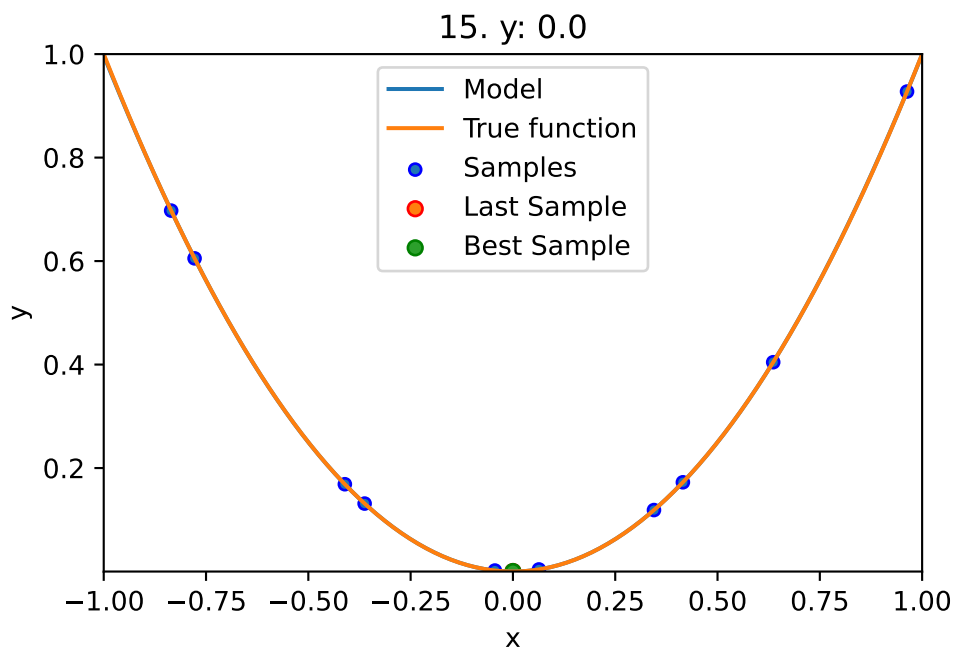
```
min y: 5.69019918867849e-10  
x0: 2.3854138401288967e-05
```

```
[['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

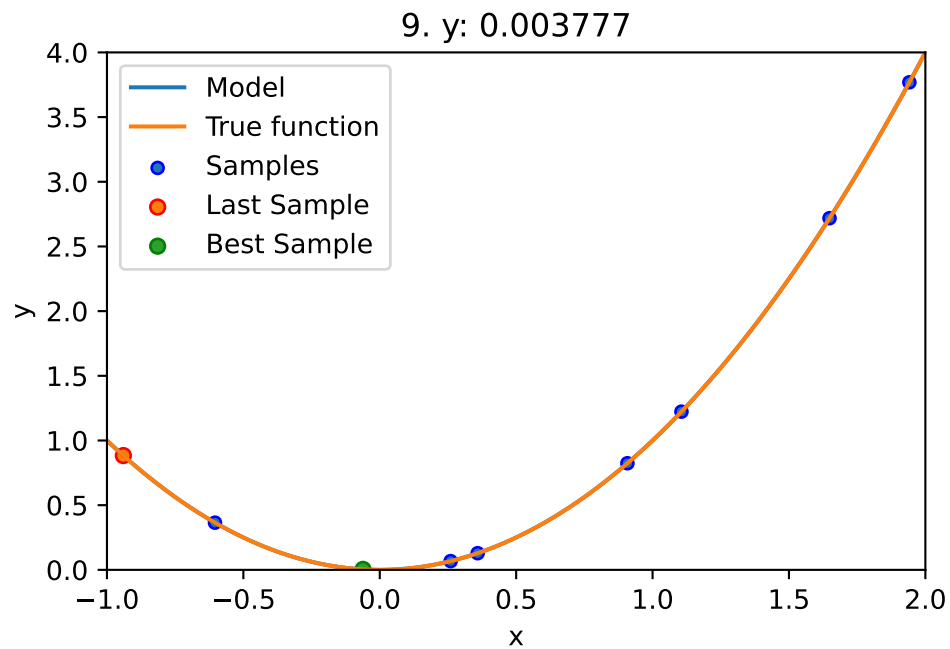
We will modify three parameters:

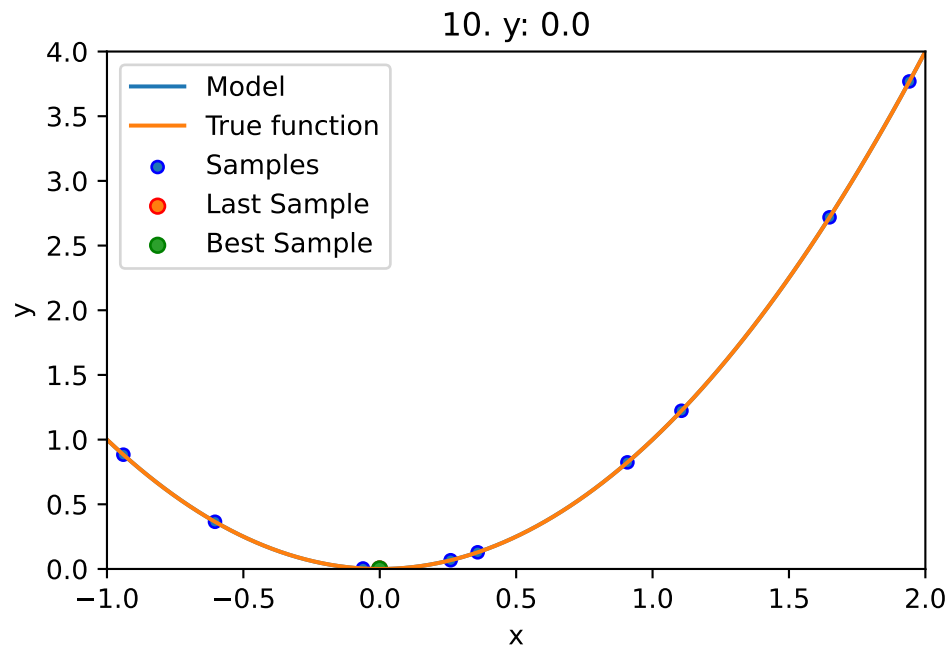
1. The number of function evaluations (`fun_evals`)
2. The size of the initial design (`init_size`)

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 0x16372eb30>

1.5 Print the Results

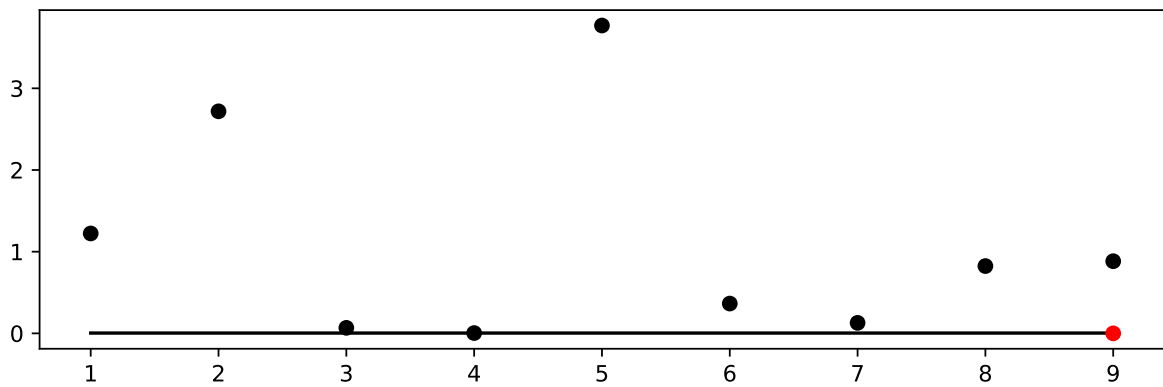
```
spot_1.print_results()
```

```
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 0x10c8fdab0>
```

2.1.2 Results

```
spot_3.print_results()
```

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

```
Pressure: 0.00514742089151478
```

```
Temp: 0.001954003740617489
```

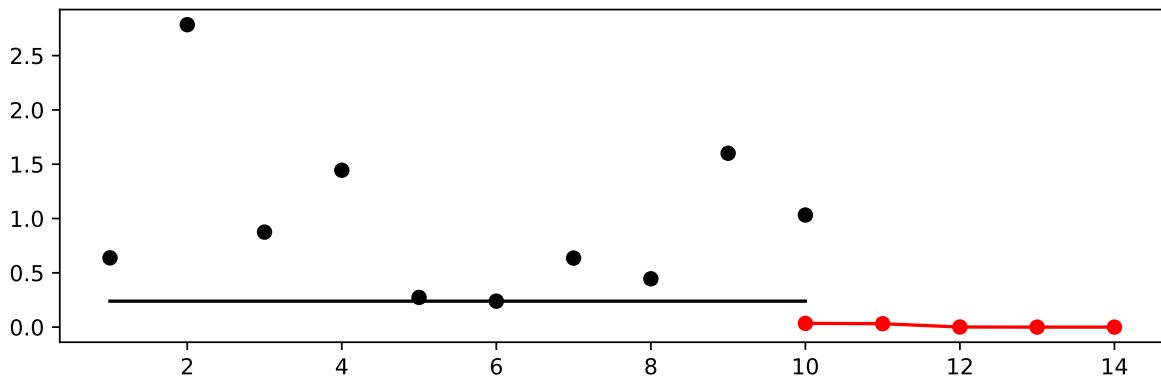
```
Lambda: 0.005476012040857559
```

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

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

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

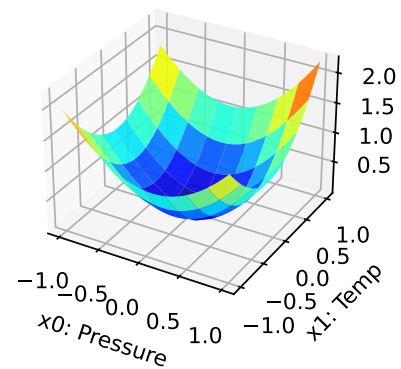
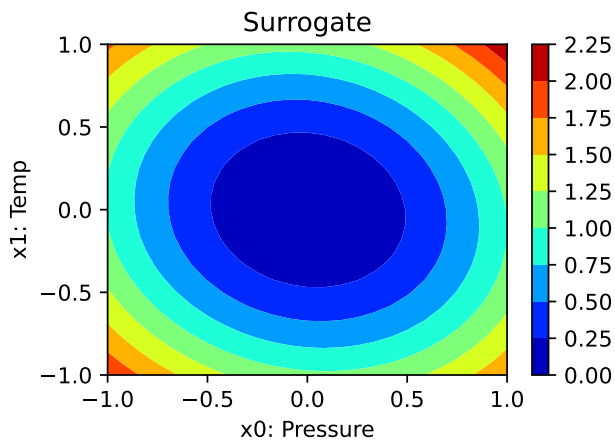
```
spot_3.plot_progress()
```



2.1.3 A Contour Plot

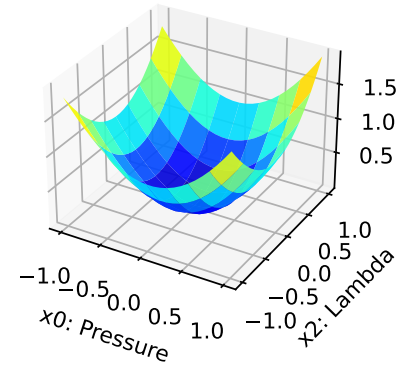
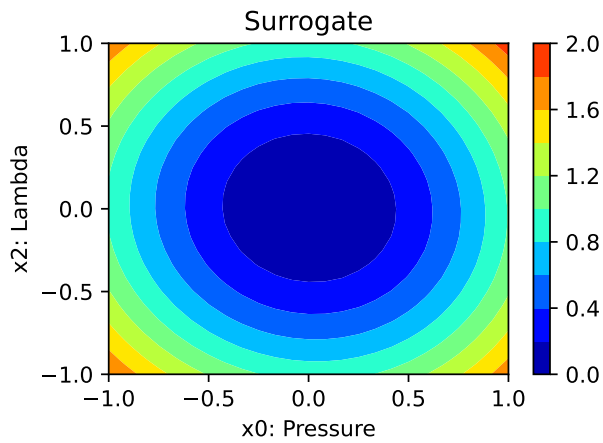
- We can select two dimensions, say $i = 0$ and $j = 1$, and generate a contour plot as follows.
 - Note: We have specified identical `min_z` and `max_z` values to generate comparable plots!

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



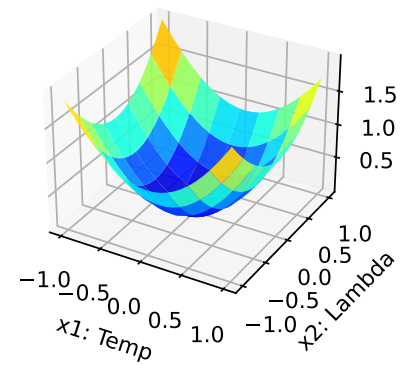
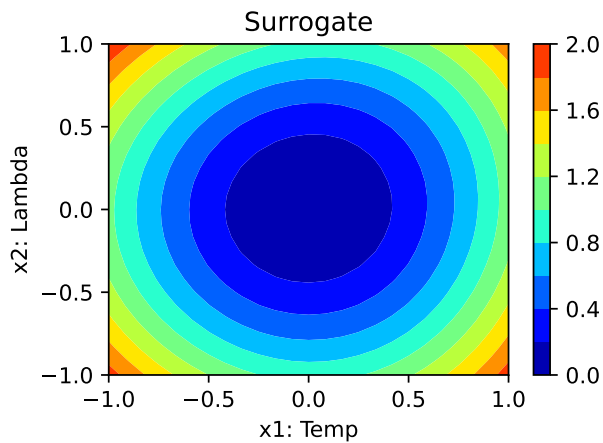
- In a similar manner, we can plot dimension $i = 0$ and $j = 2$:

```
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 0x1547e2950>
```

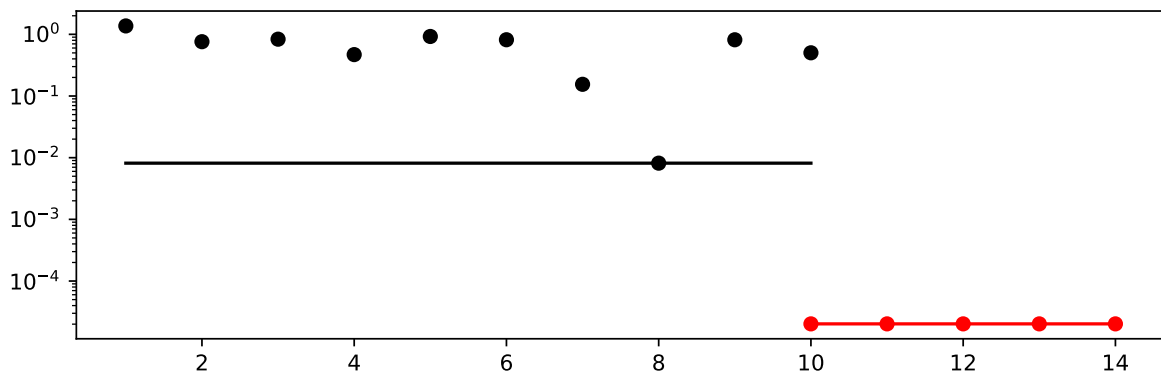
3.1.2 Results

```
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 0x1593c7370>
```

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])
```

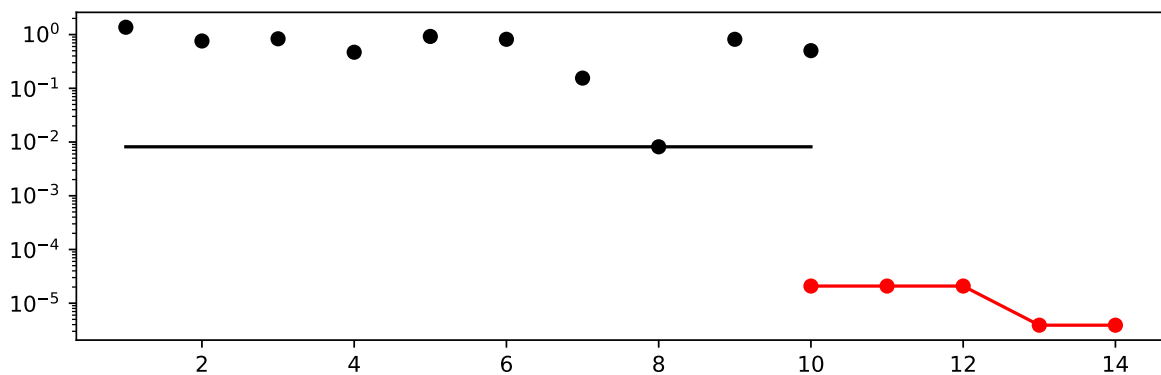
- Since the surrogate from the isotropic setting was stored as `spot_2`, we can also take a look at the `theta` value from this model:

```
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()
```

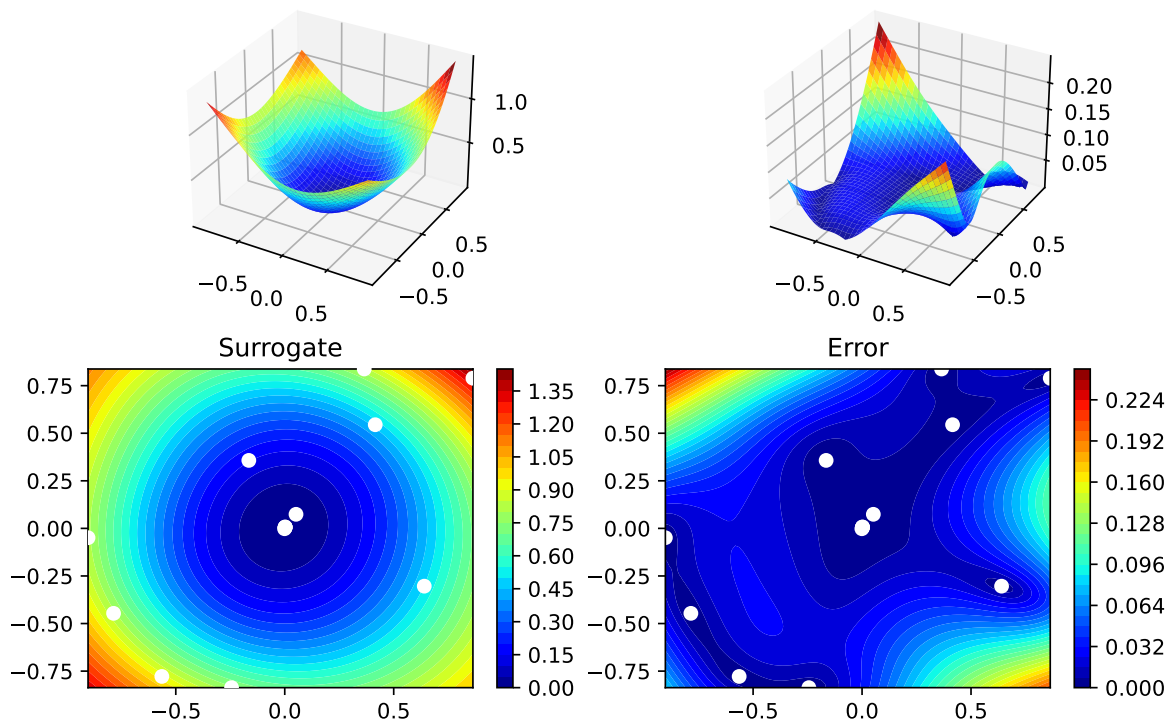
min y: 3.898914658670152e-06

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 0x15978e620>
```

4.1.3 Print the Results

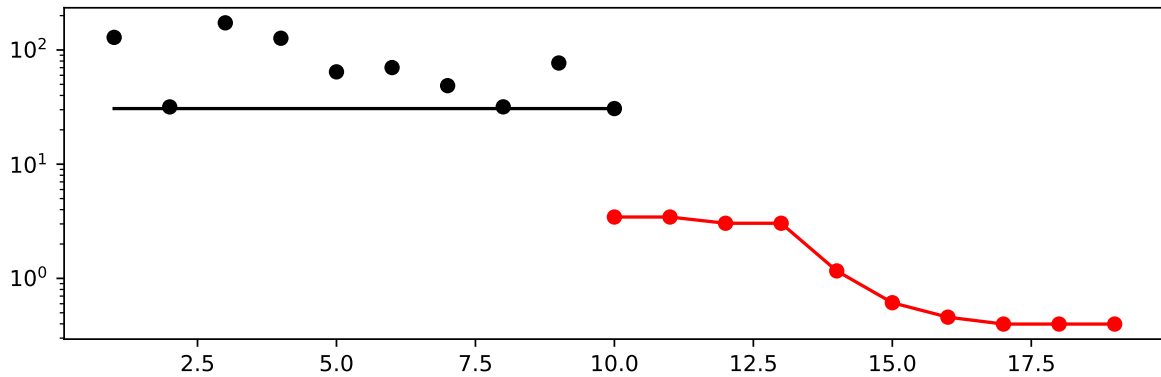
```
spot_2.print_results()
```

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

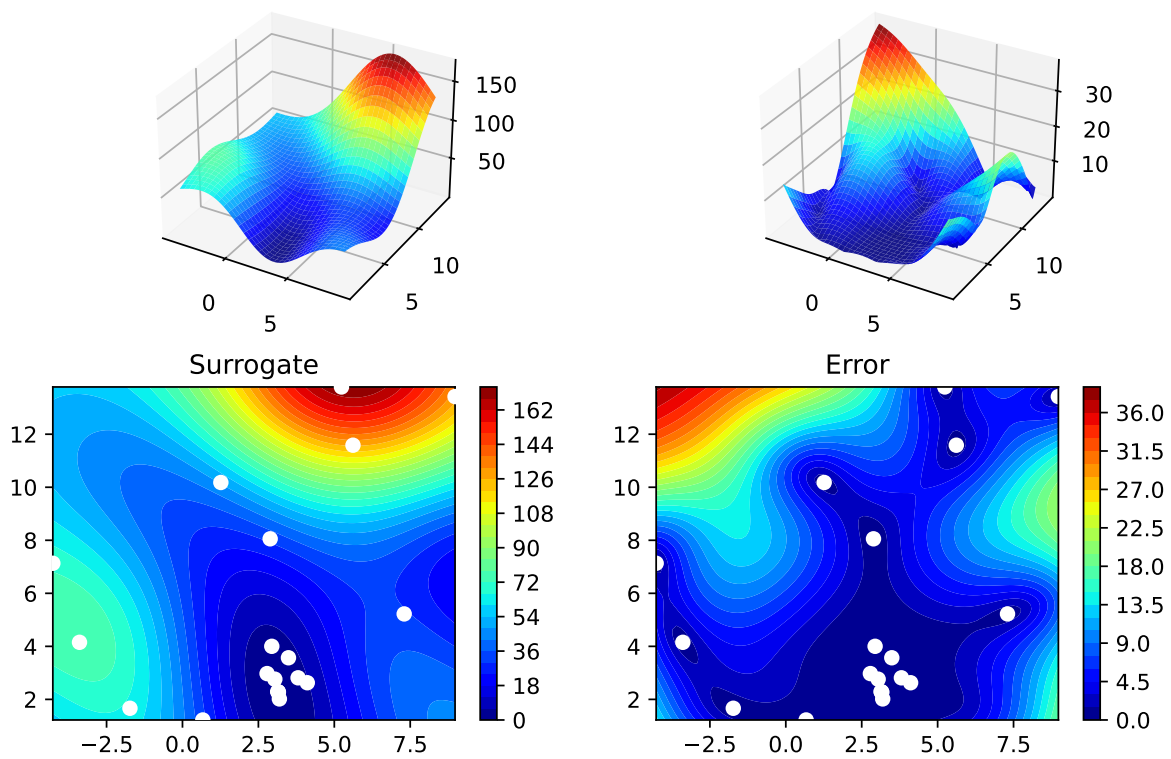
```
[['x0', 3.135563584477711], ['x1', 2.2926607128616965]]
```

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

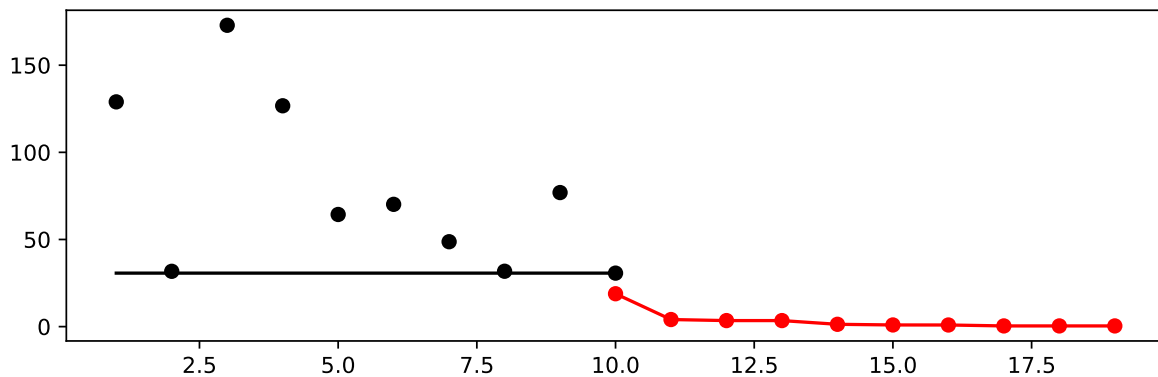
- Similar to the Spot run with the internal Kriging model, we can call the run with the scikit-learn surrogate:

```
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 0x15e658880>

```
spot_2_GP.plot_progress()
```



```
spot_2_GP.print_results()
```

min y: 0.3982852142065738

x0: 3.150695392291088

x1: 2.2681225819692425

```
[['x0', 3.150695392291088], ['x1', 2.2681225819692425]]
```

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

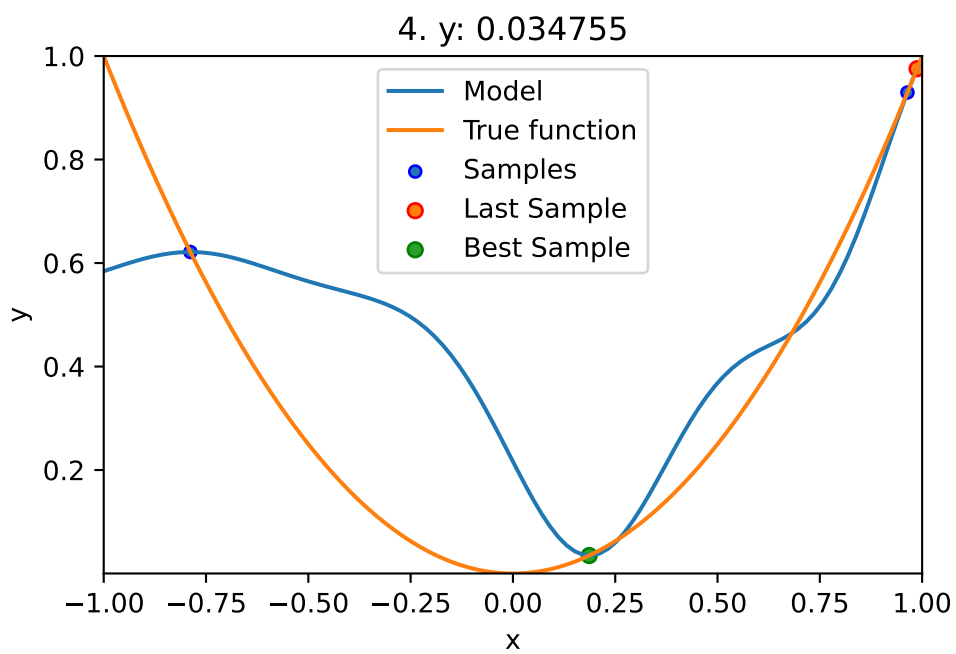
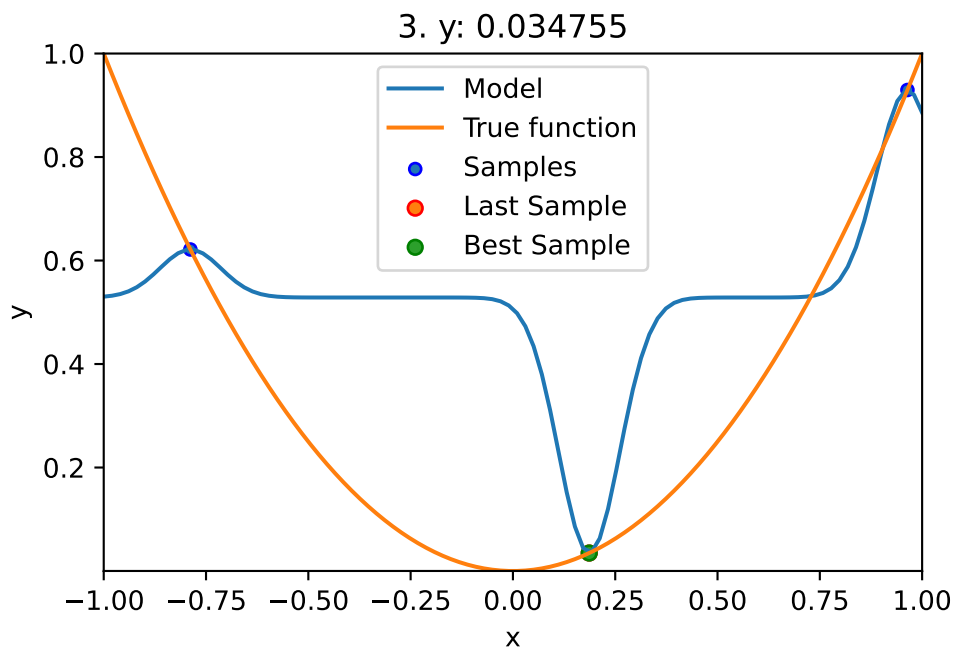
- In this example, we will use an one-dimensional function, which allows us to visualize the optimization process.

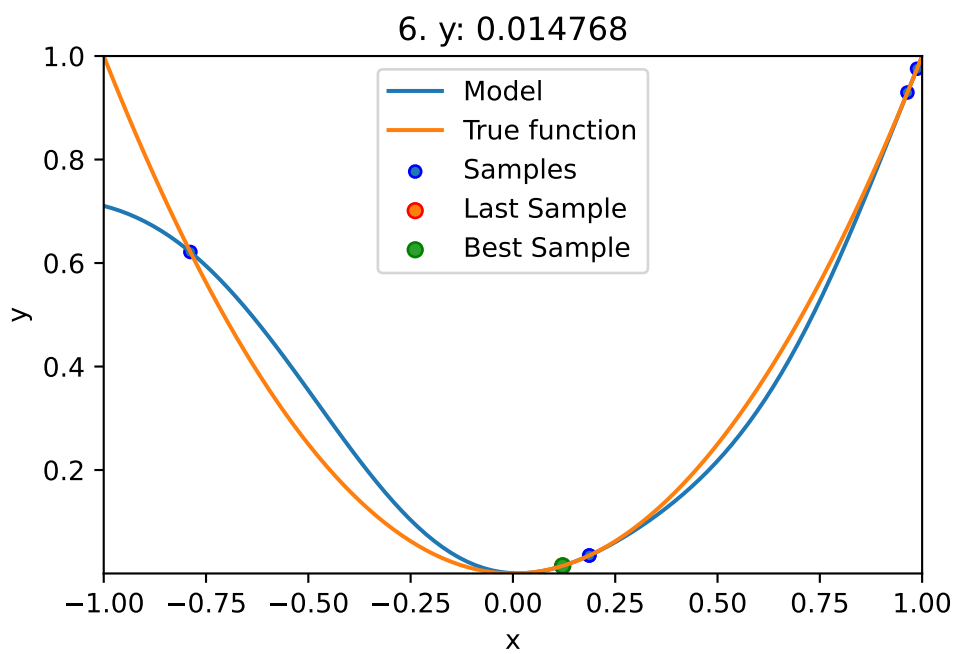
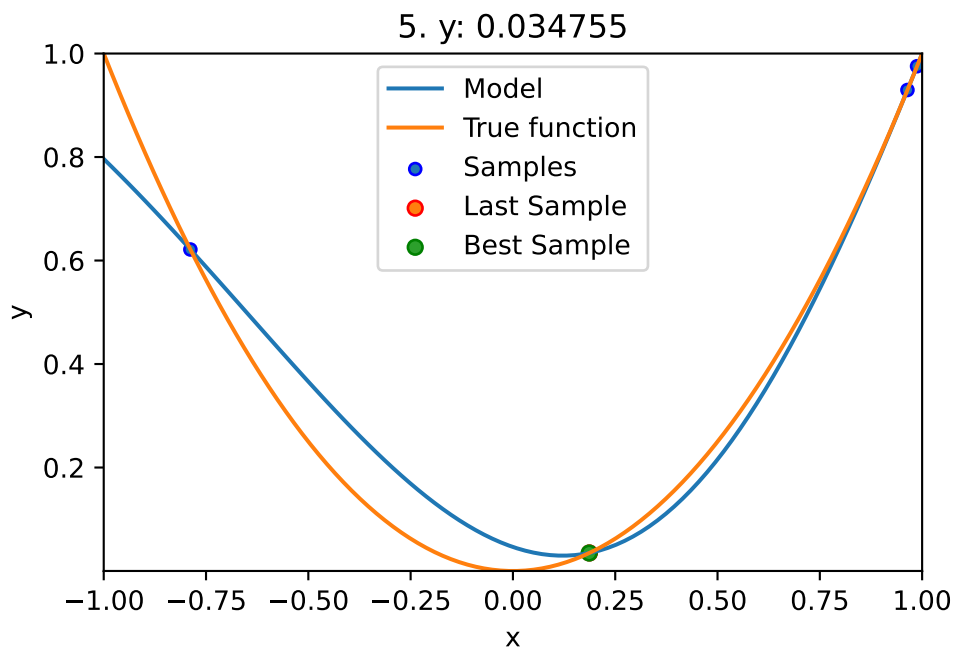
– `show_models= True` is added to the argument list.

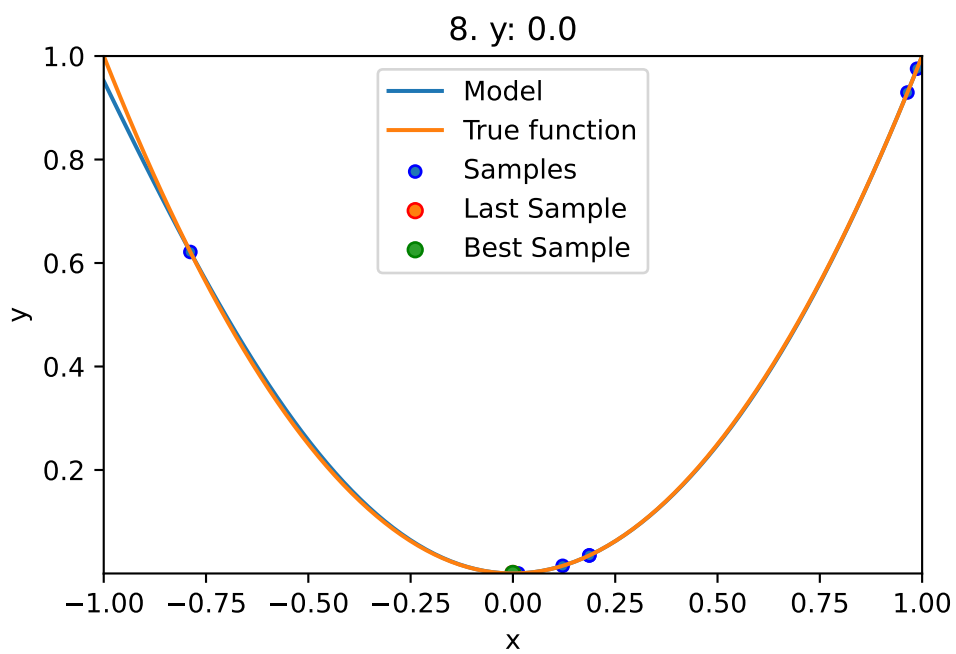
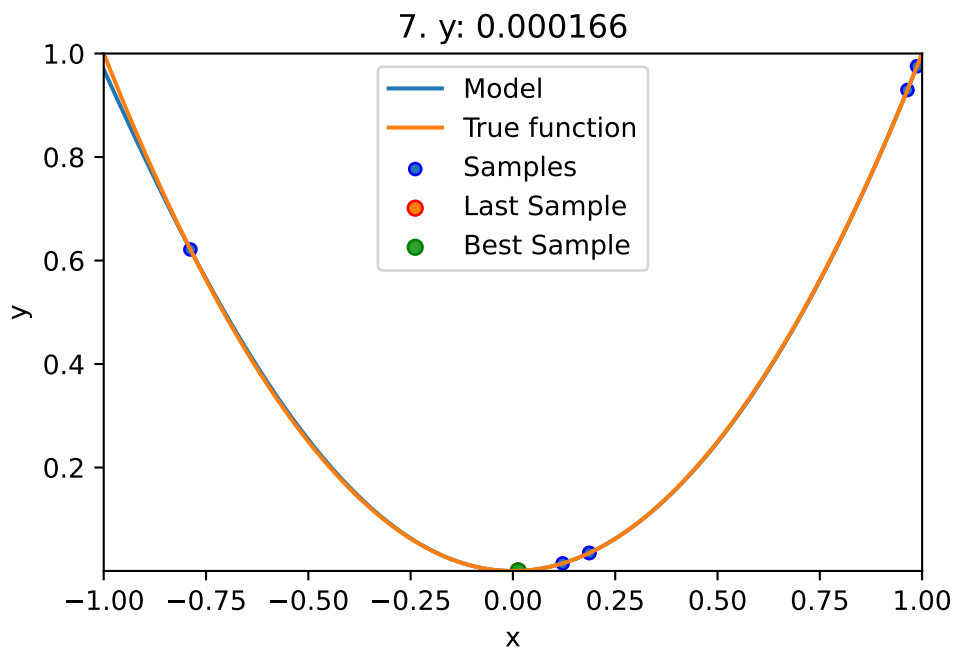
```
from spotPython.fun.objectivefunctions import analytical
lower = np.array([-1])
upper = np.array([1])
fun = analytical(seed=123).fun_sphere

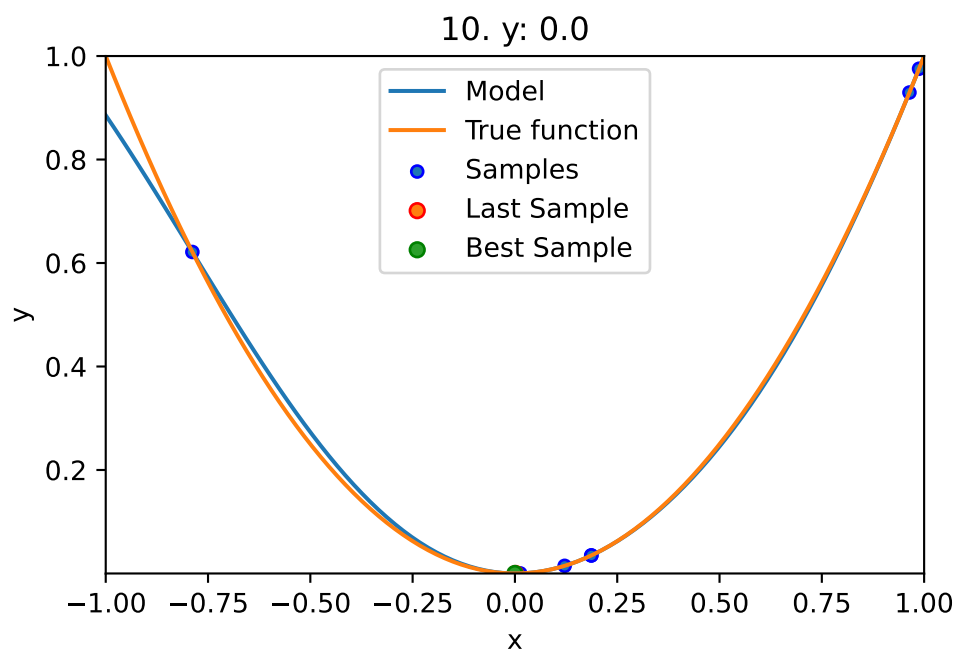
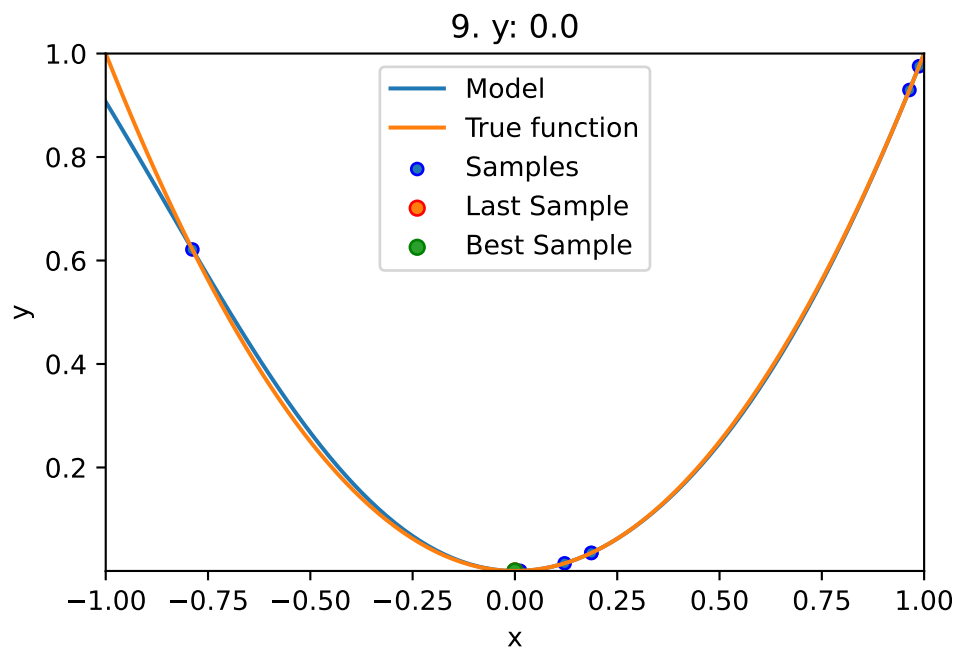
spot_1 = spot.Spot(fun=fun,
                   lower = lower,
                   upper = upper,
                   fun_evals = 10,
                   max_time = inf,
                   seed=123,
                   show_models= True,
                   tolerance_x = np.sqrt(np.spacing(1)),
          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

spot_1.run()
```









<spotPython.spot.spot.Spot at 0x15e6156f0>

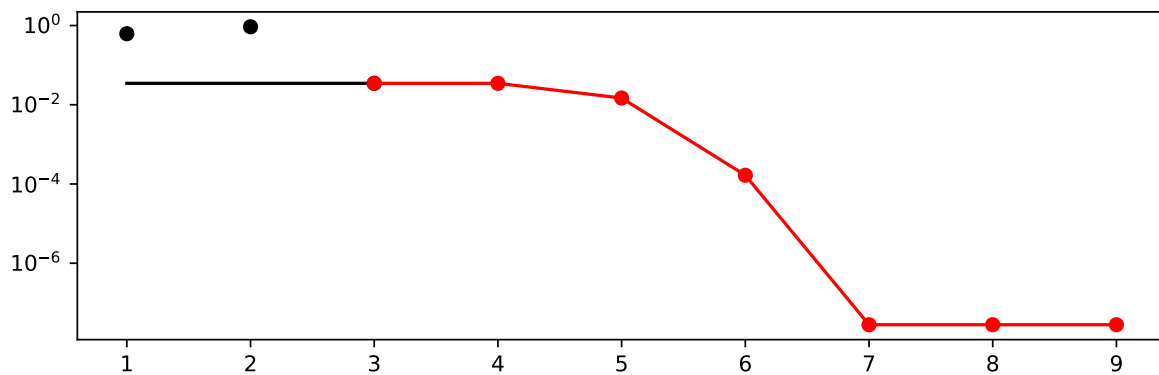
4.3.1 Results

```
spot_1.print_results()
```

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

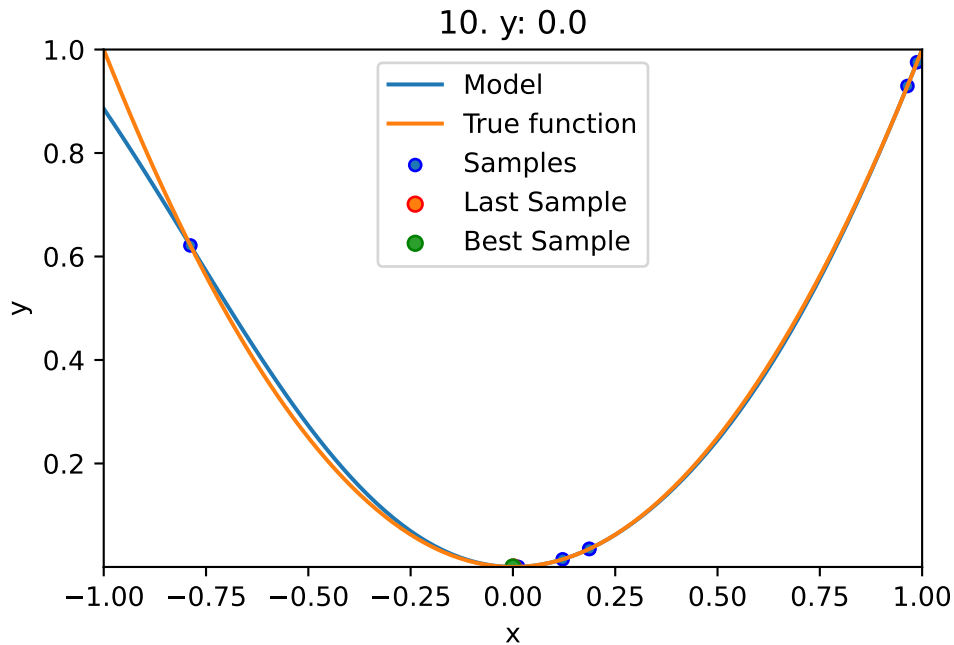
```
[['x0', -0.0001673274293477195]]
```

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



- The method `plot_model` plots the final surrogate:

```
spot_1.plot_model()
```

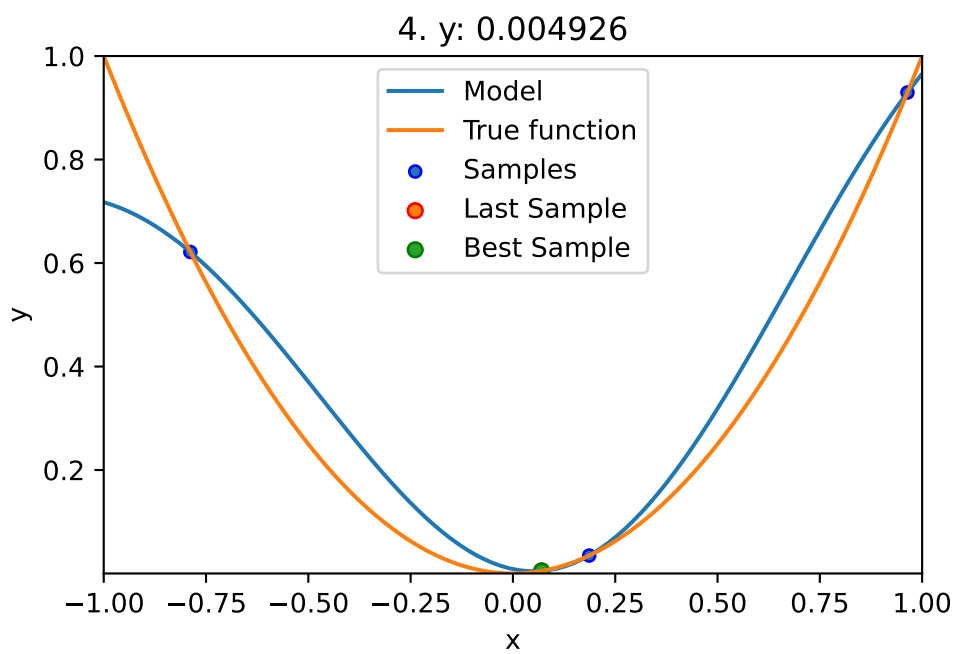
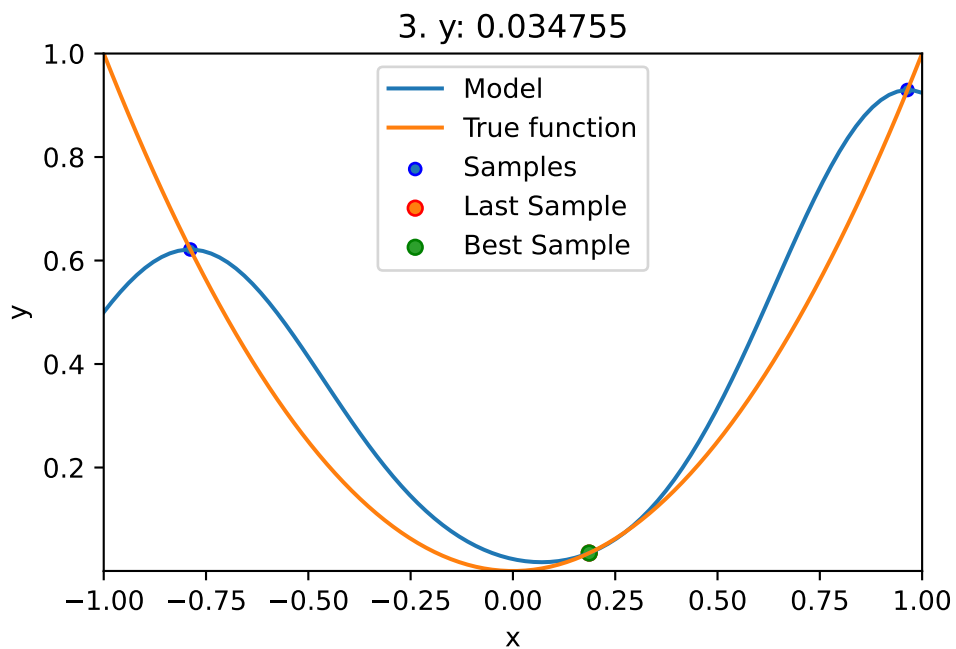


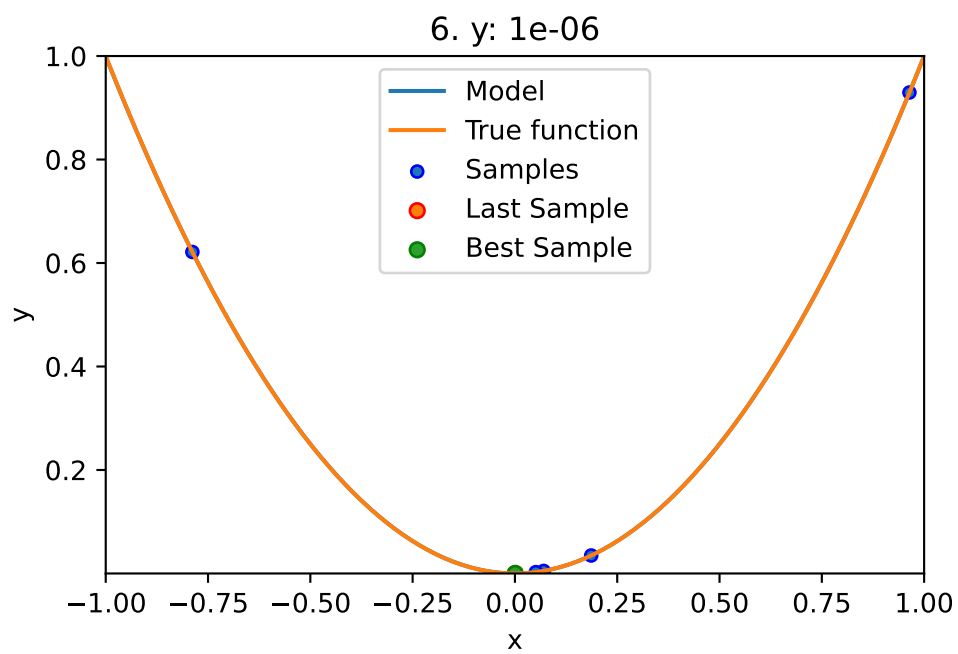
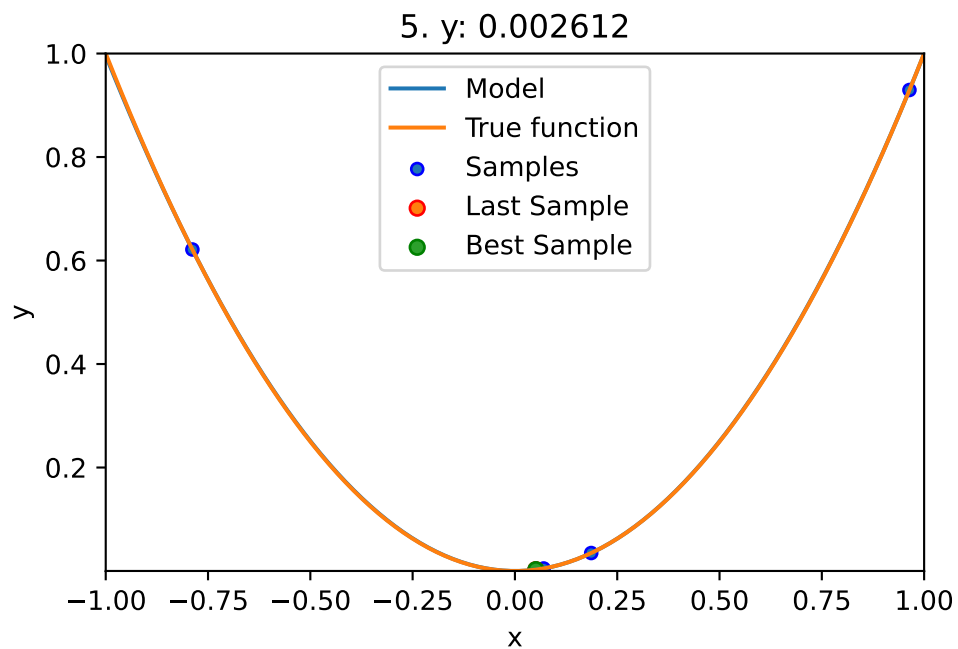
4.4 Example: Sklearn Model GaussianProcess

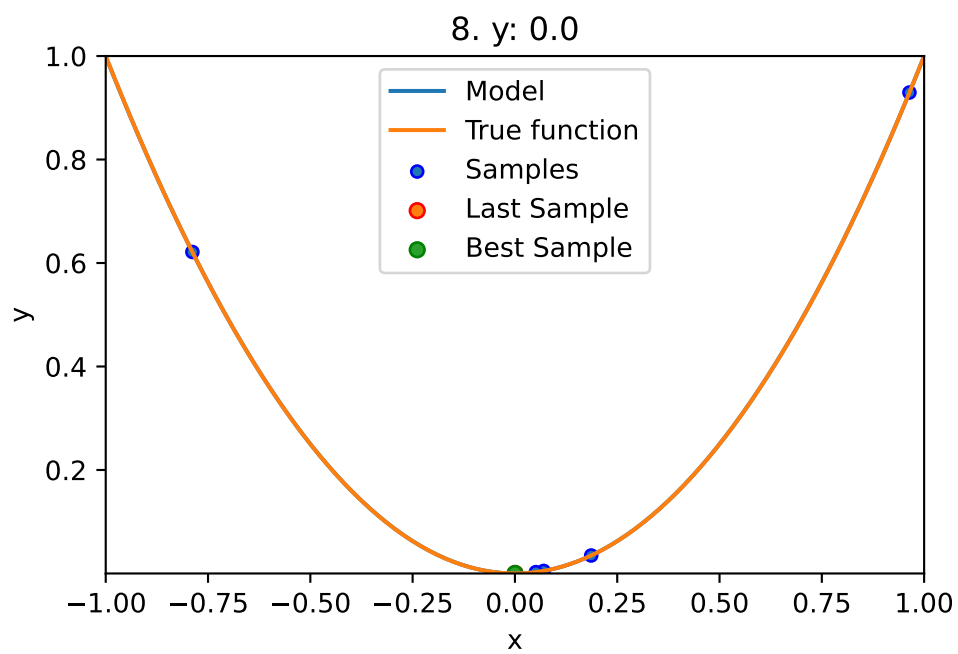
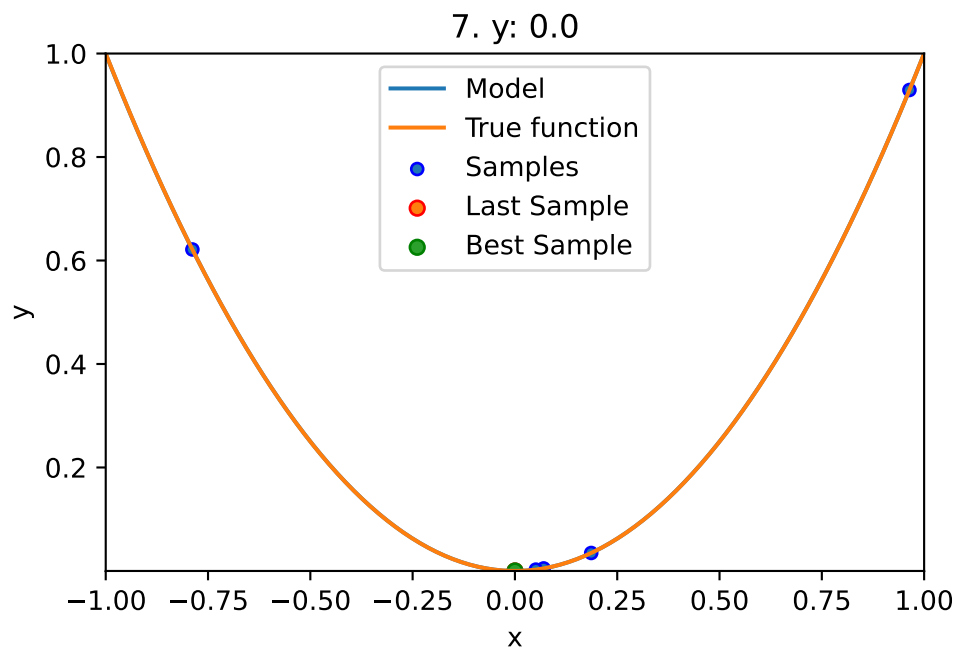
- This example visualizes the search process on the `GaussianProcessRegression` surrogate from `sklearn`.
- Therefore `surrogate = S_GP` is added to the argument list.

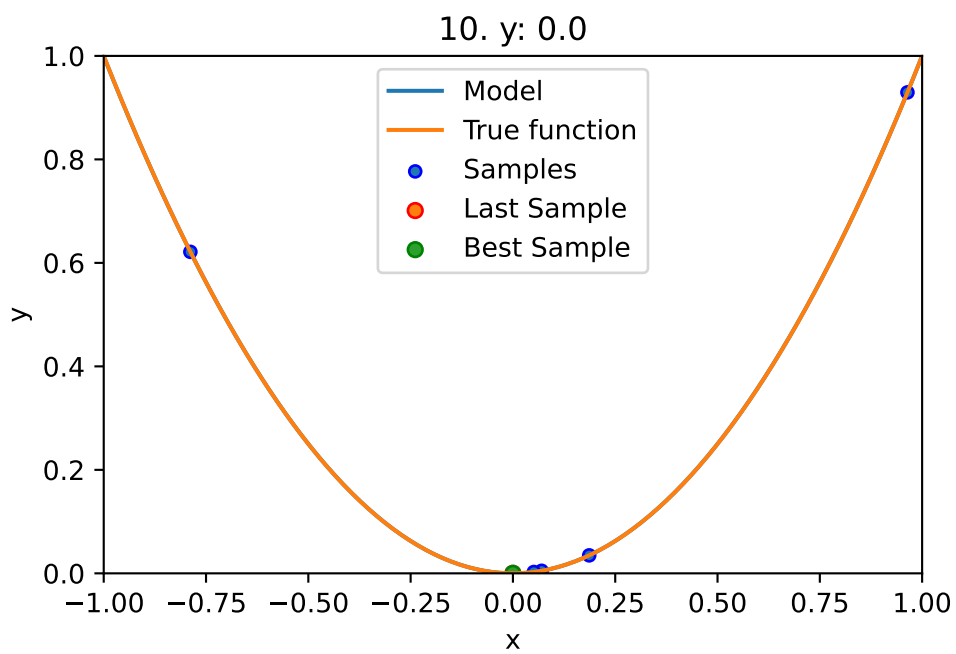
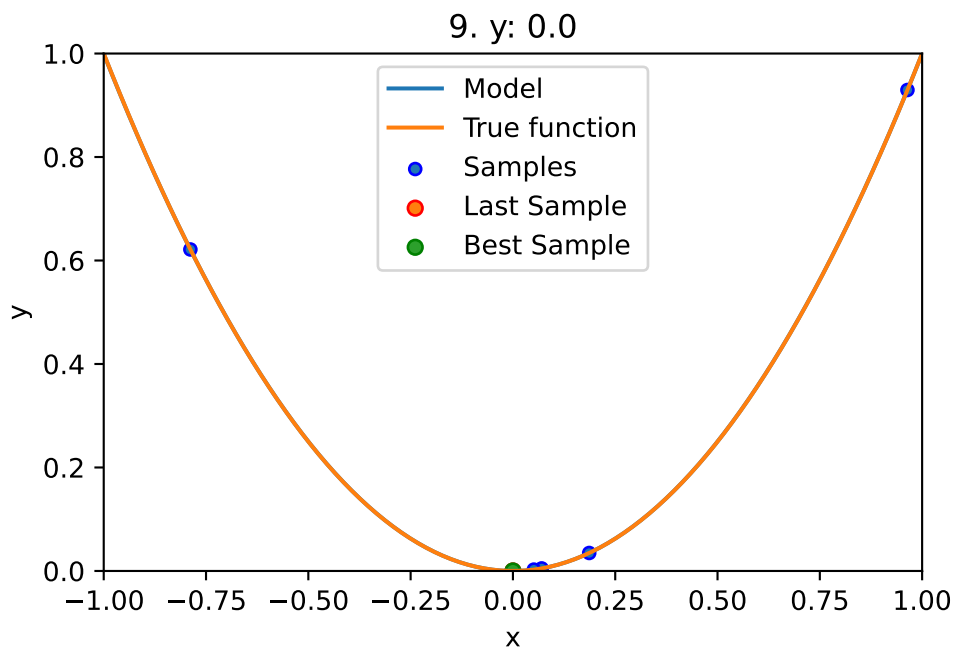
```
fun = analytical(seed=123).fun_sphere
spot_1_GP = spot.Spot(fun=fun,
                      lower = lower,
                      upper = upper,
                      fun_evals = 10,
                      max_time = inf,
                      seed=123,
                      show_models= True,
    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

spot_1_GP.run()
```







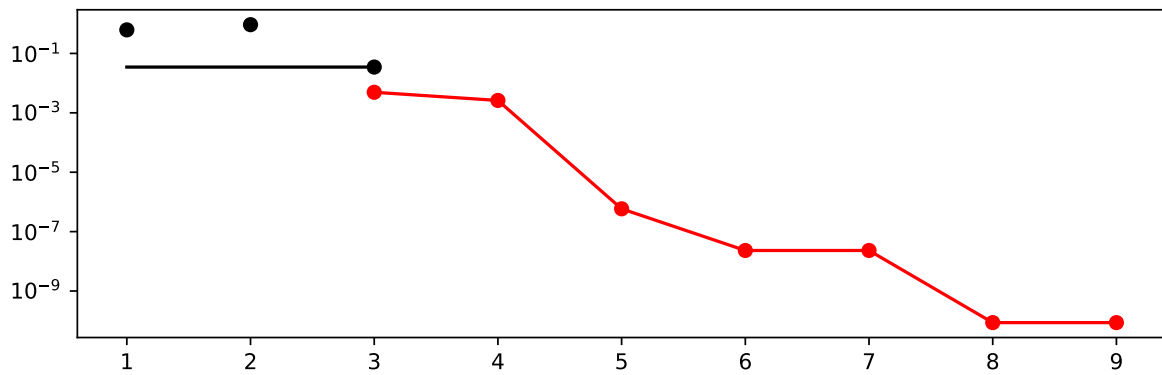
<spotPython.spot.spot.Spot at 0x15e46d930>

```
spot_1_GP.print_results()
```

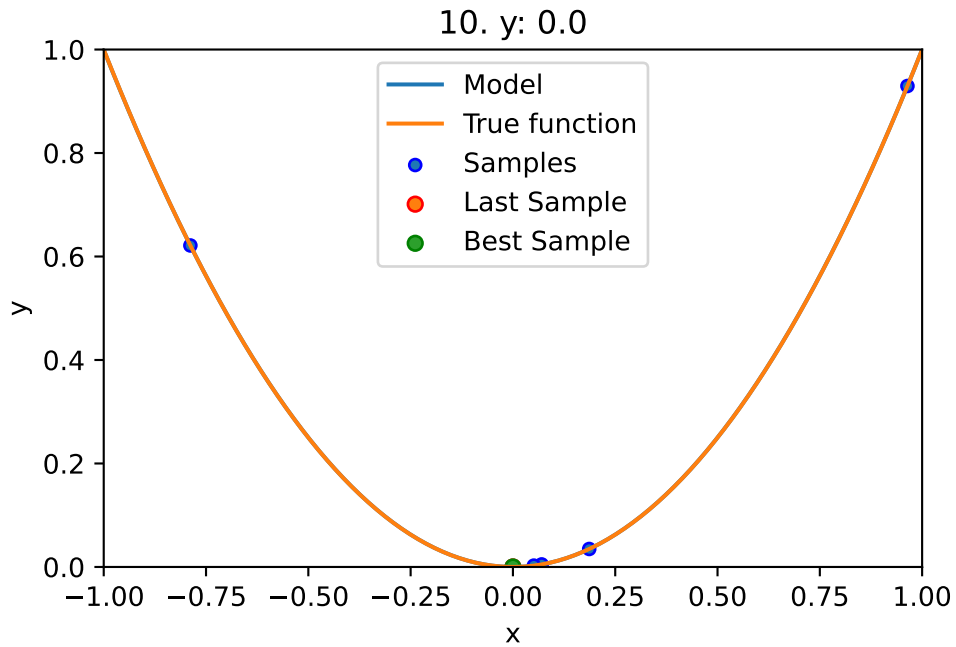
```
min y: 8.62352843114662e-11  
x0: -9.28629551066873e-06
```

```
[['x0', -9.28629551066873e-06]]
```

```
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 0x10d19b610>

5.3 Print the Results

```
spot_de.print_results()
```

```

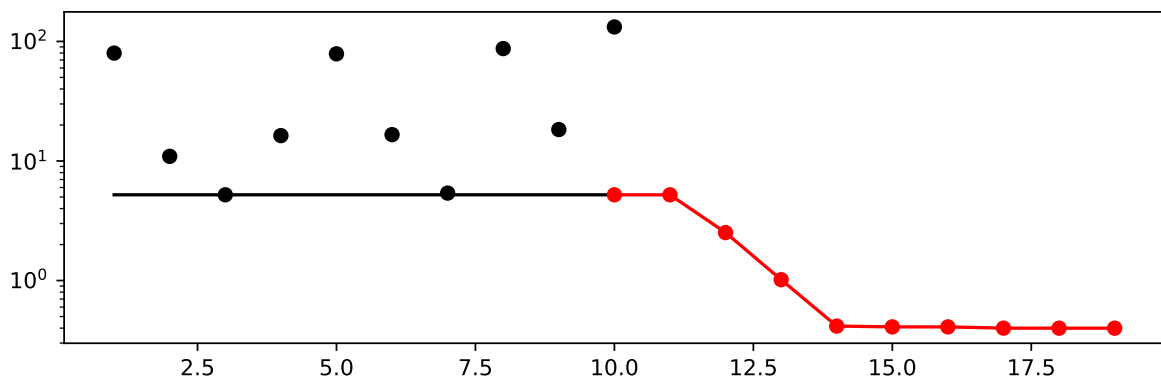
min y: 0.3995745460887754
x0: -3.1573237022835743
x1: 12.290496562954784

```

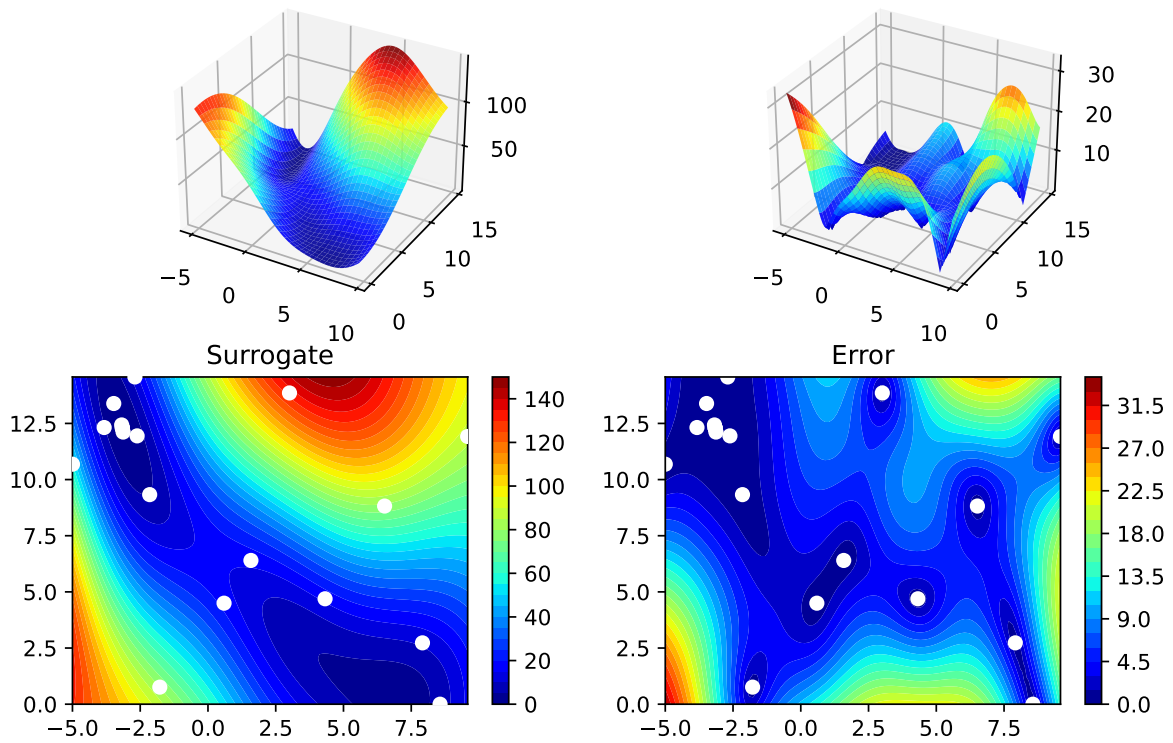
```
[['x0', -3.1573237022835743], ['x1', 12.290496562954784]]
```

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

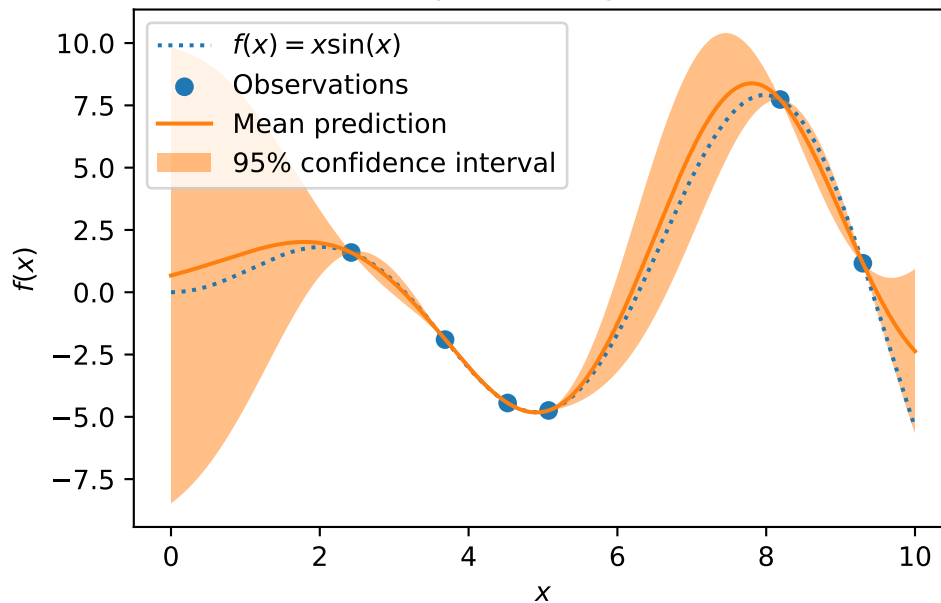
- The model building with `sklearn` consists of three steps:
 1. Instantiating the model, then
 2. fitting the model (using `fit`), and
 3. making predictions (using `predict`)

```
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")
```

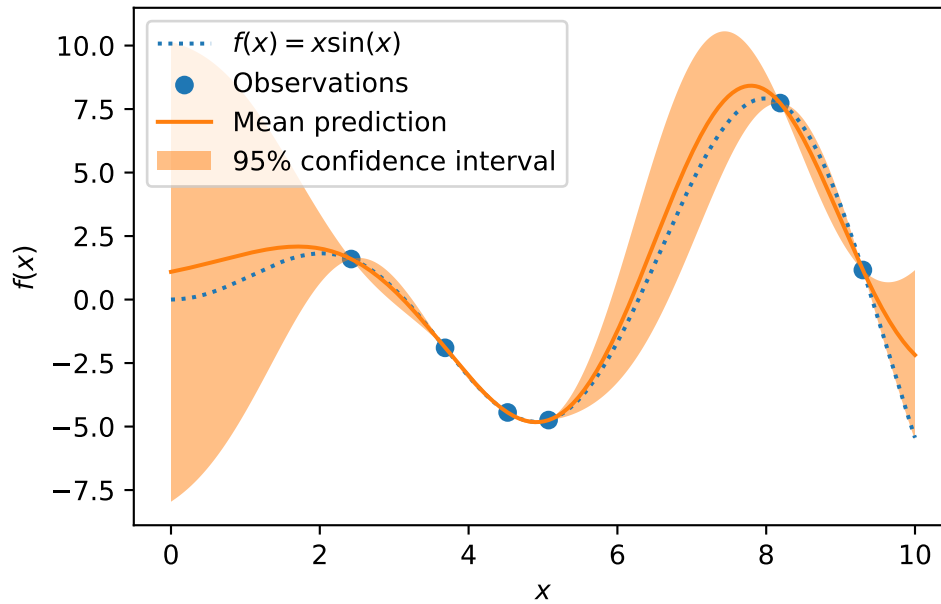
```
plt.plot(X, y, label=r"$f(x) = x \sin(x)$", linestyle="dotted")
plt.scatter(X_train, y_train, label="Observations")
plt.plot(X, S_mean_prediction, label="Mean prediction")
plt.fill_between(
    X.ravel(),
    S_mean_prediction - 1.96 * S_std_prediction,
    S_mean_prediction + 1.96 * S_std_prediction,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

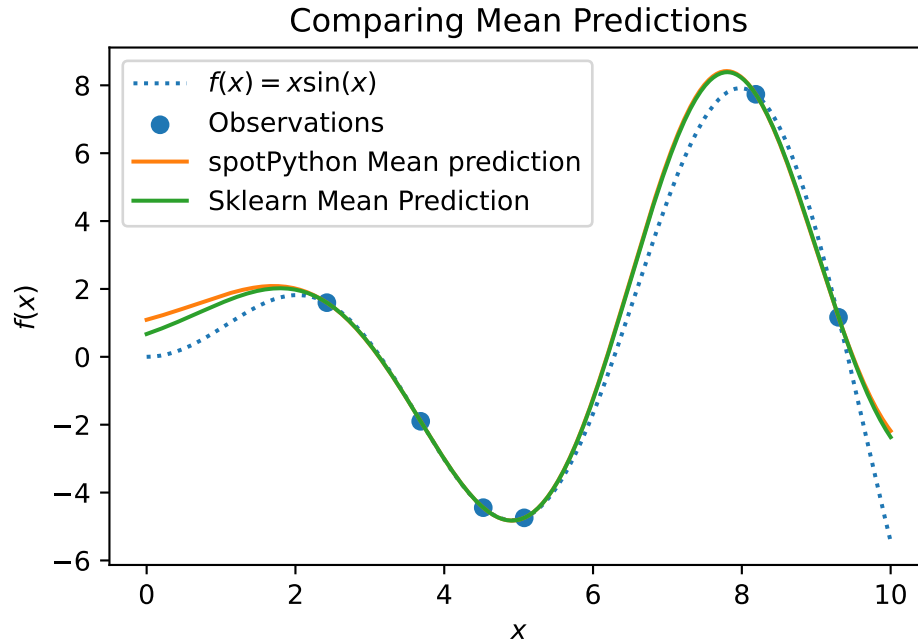


6.1.5 Visualizing the Differences Between the spotPython and the sklearn Model Fits

```

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 0x15cc668c0>
```

7.1.2 Results

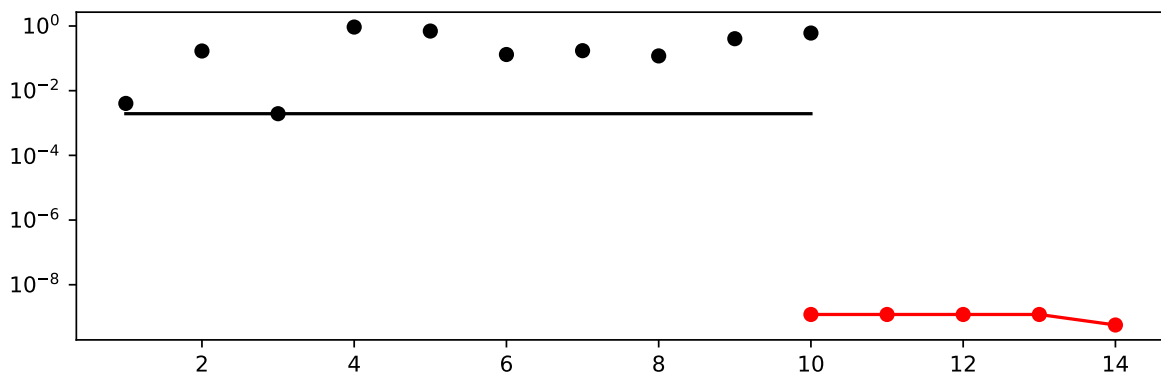
```
spot_1.print_results()
```

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

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

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

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

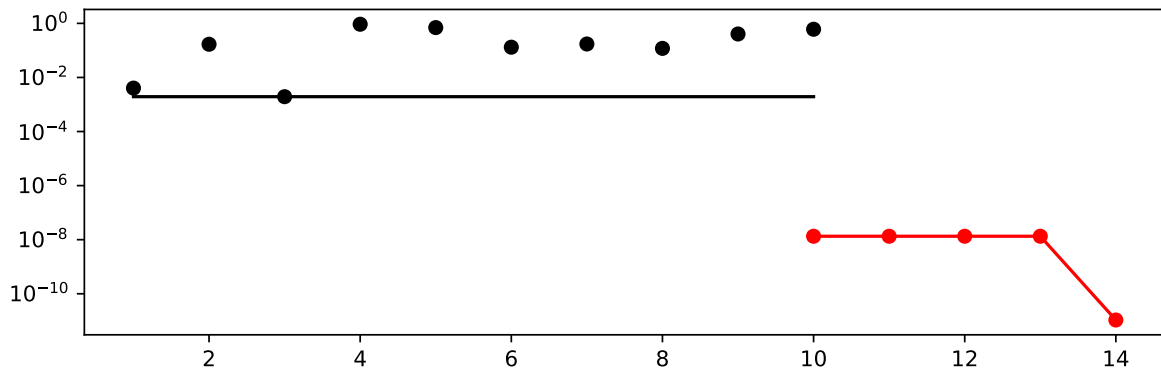


7.2 Same, but with EI as infill_criterion

```
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 0x1616c6320>
```

```
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,
```

```

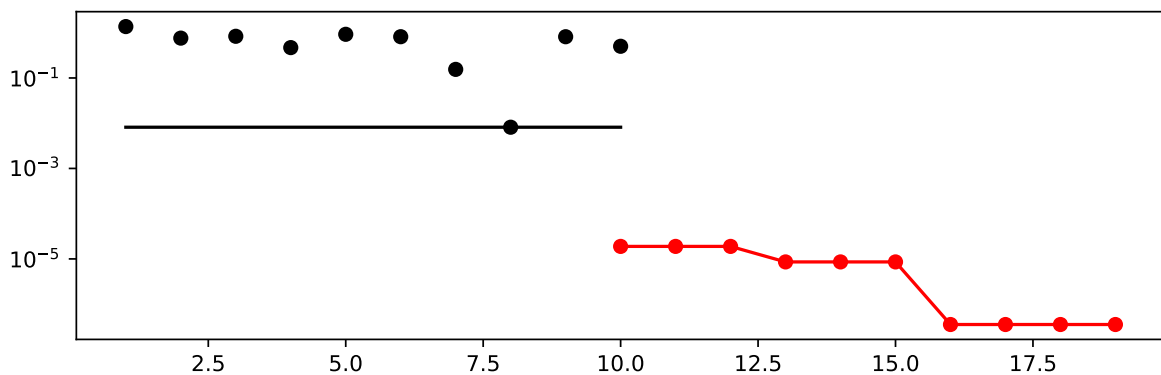
design_control={"init_size": 10,
               "repeats": 1},
surrogate_control={"noise": False,
                  "cod_type": "norm",
                  "min_theta": -4,
                  "max_theta": 3,
                  "n_theta": 2,
                  "model_optimizer": differential_evolution,
  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

spot_2_ei_noniso.run()

```

<spotPython.spot.spot.Spot at 0x1617e2650>

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



```
spot_2_ei_noniso.print_results()
```

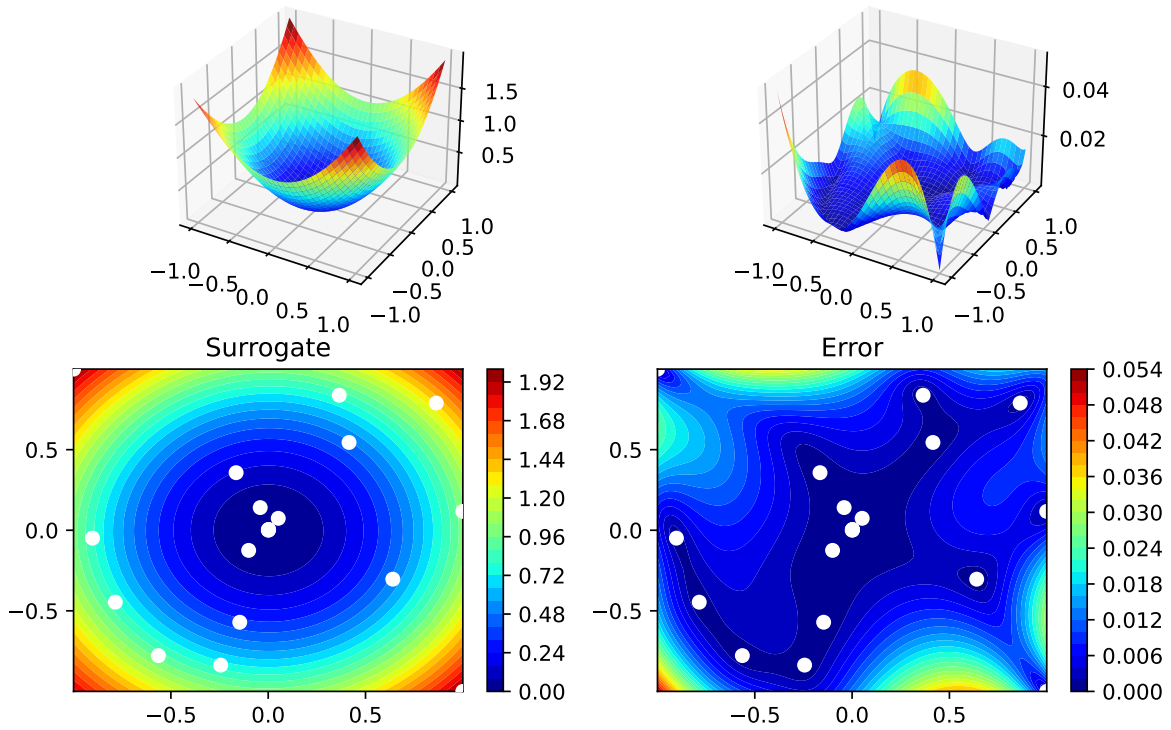
```

min y: 3.546757229720085e-07
x0: 0.0005304763214324646
x1: 0.0002706854177296757

```

```
[['x0', 0.0005304763214324646], ['x1', 0.0002706854177296757]]
```

```
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:

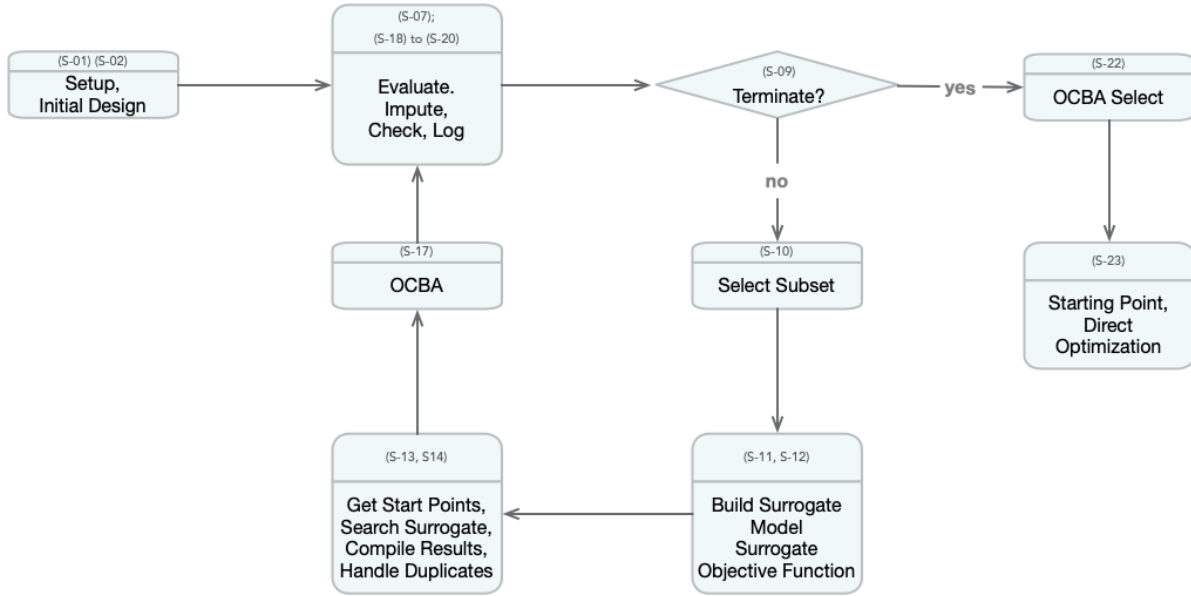


Figure 7.1: Visual representation of the model based search with SPOT. Taken from: Bartz-Beielstein, T., and Zaefferer, M. Hyperparameter tuning approaches. In Hyperparameter Tuning for Machine and Deep Learning with R - A Practical Guide, E. Bartz, T. Bartz-Beielstein, M. Zaefferer, and O. Mersmann,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

7.4.2 spot: The Initial Model

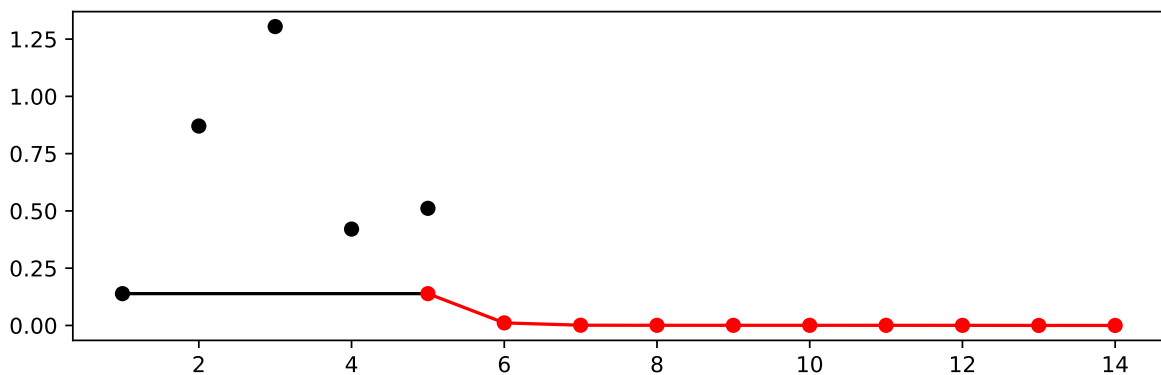
7.4.2.1 Example: Modifying the initial design size

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 0x161b2ed40>

```
spot_ei.plot_progress()
```



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

(5.69019918867849e-10, 1.8944758020557087e-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)

```

```

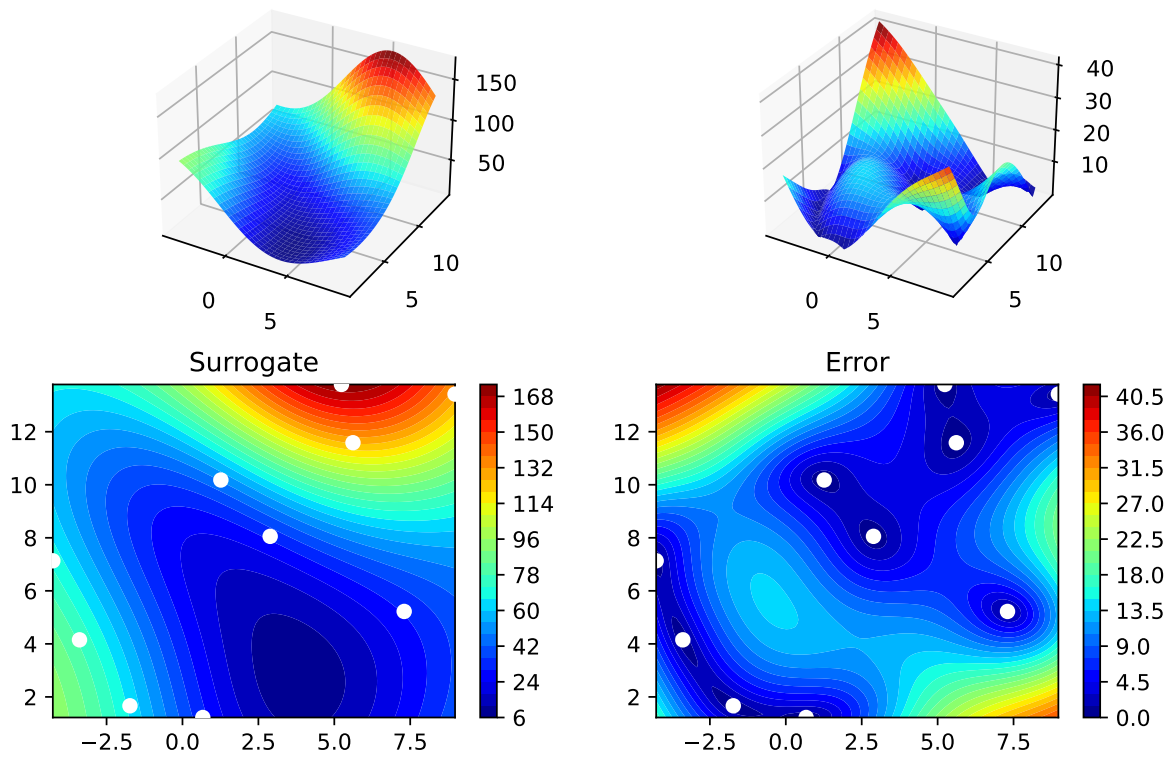
[[ 8.97647221 13.41926847]
 [ 0.66946019  1.22344228]
 [ 5.23614115 13.78185824]
 [ 5.6149825  11.5851384 ]
 [-1.72963184  1.66516096]
 [-4.26945568  7.1325531 ]
 [ 1.26363761 10.17935555]
 [ 2.88779942  8.05508969]
 [-3.39111089  4.15213772]
 [ 7.30131231  5.22275244]]
[128.95676449  31.73474356 172.89678121 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]

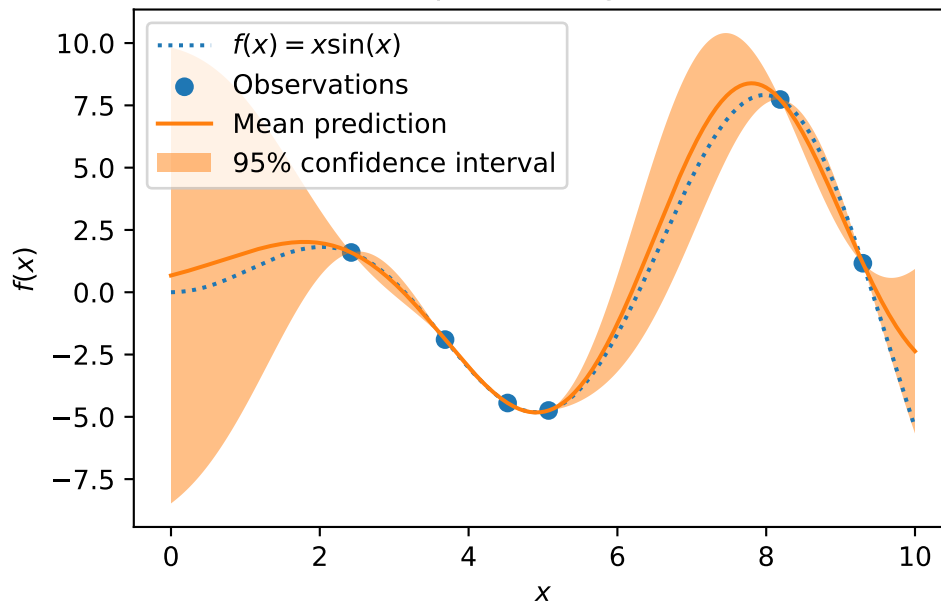
kernel = 1 * RBF(length_scale=1.0, length_scale_bounds=(1e-2, 1e2))
gaussian_process = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=9)
gaussian_process.fit(X_train, y_train)
gaussian_process.kernel_

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(
```

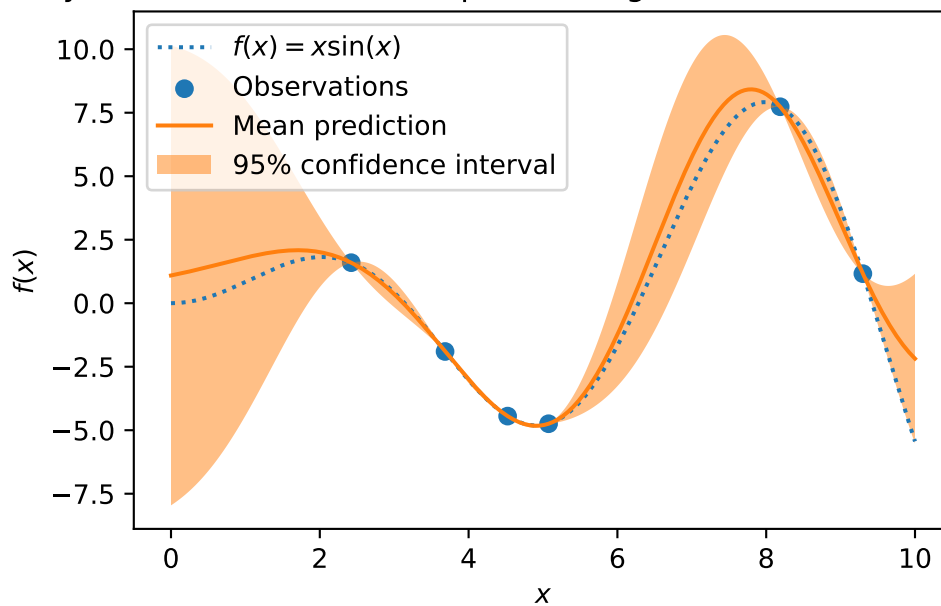


```

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)
```

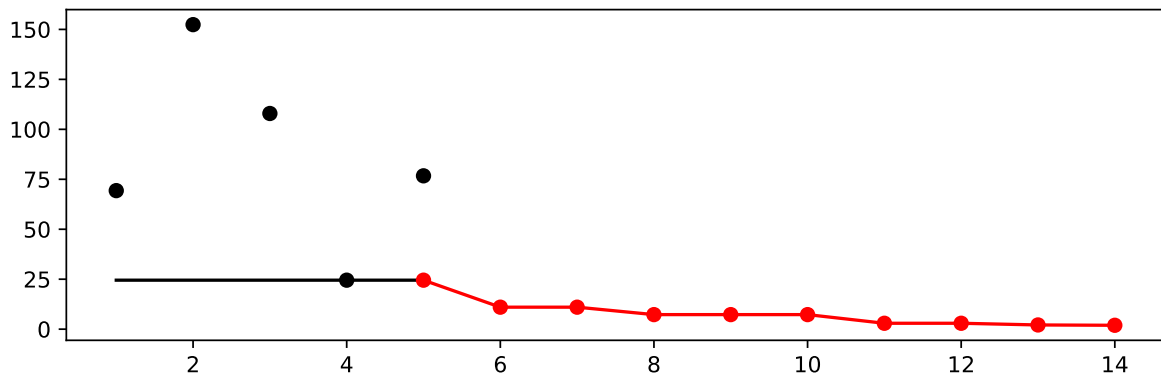
```
spot_GP.run()
```

```
<spotPython.spot.spot.Spot at 0x16222ca30>
```

```
spot_GP.y
```

```
array([ 69.32459936, 152.38491454, 107.92560483,  24.51465459,  
       76.73500031,  86.30426444,  11.00311552,  16.11743403,  
        7.28122228,  21.82314306,  10.96088904,   2.95178528,  
        3.02904722,   2.10495286,   1.94315988])
```

```
spot_GP.plot_progress()
```



```
spot_GP.print_results()
```

```
min y: 1.9431598813189215  
x0: 10.0  
x1: 2.9985727218884697
```

```
[['x0', 10.0], ['x1', 2.9985727218884697]]
```

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])

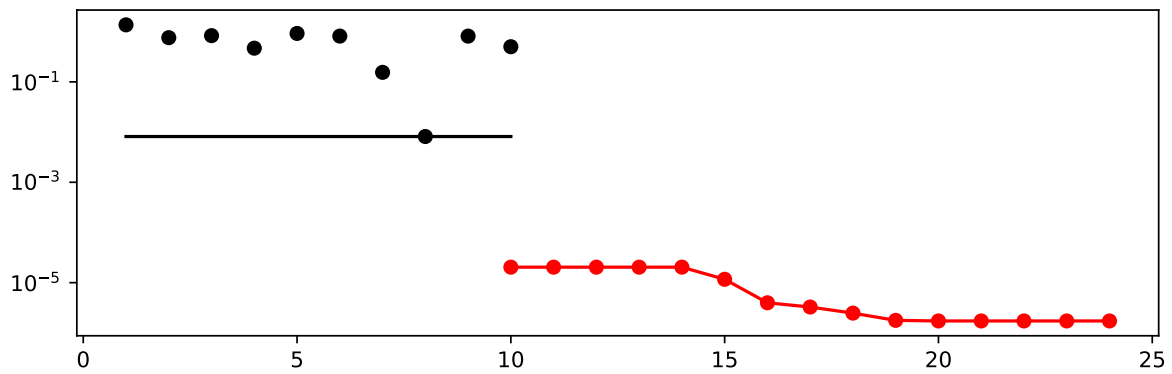
design_control={"init_size": 10}
surrogate_control={
    "n_points": 1,
}
spot_S_K = spot.Spot(fun=fun,
                    lower = lower,
                    upper= upper,
                    surrogate=S_K,
                    fun_evals = 25,
                    noise = False,
                    log_level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

```

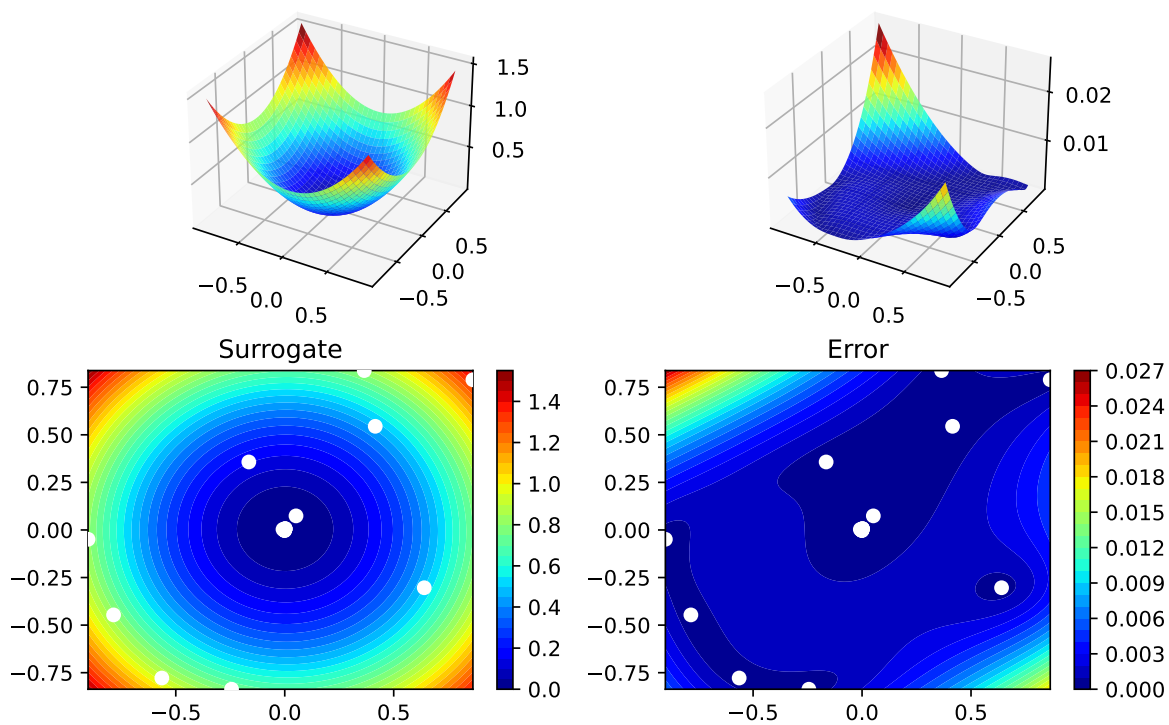
```
design_control=design_control,  
surrogate_control=surrogate_control)  
  
spot_S_K.run()
```

<spotPython.spot.spot.Spot at 0x1632a9180>

```
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)

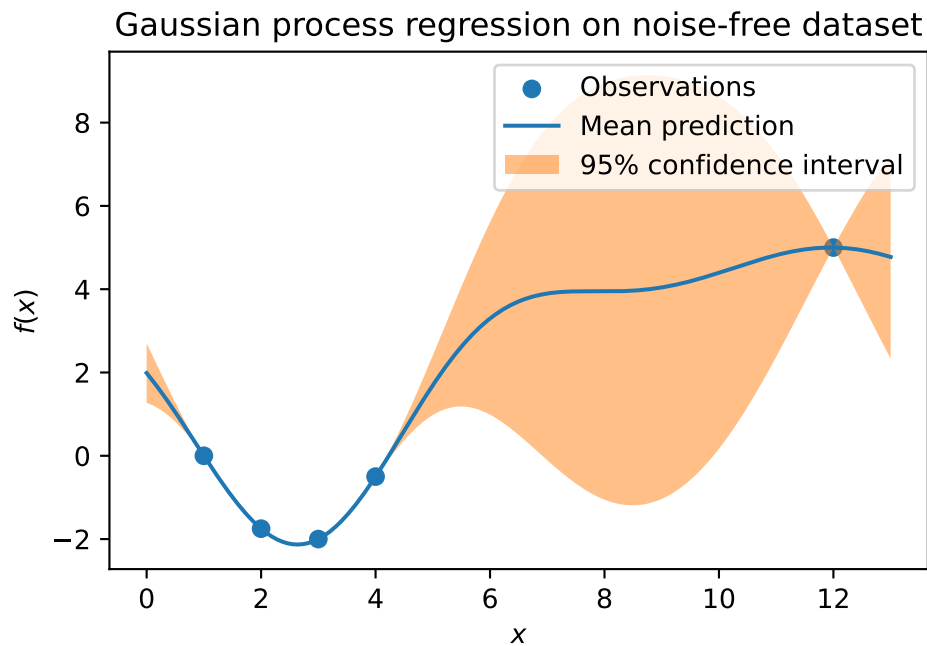
X = np.linspace(start=0, stop=13, num=1000).reshape(-1, 1)
mean_prediction, std_prediction, ei = S.predict(X, return_val="all")

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",
    )

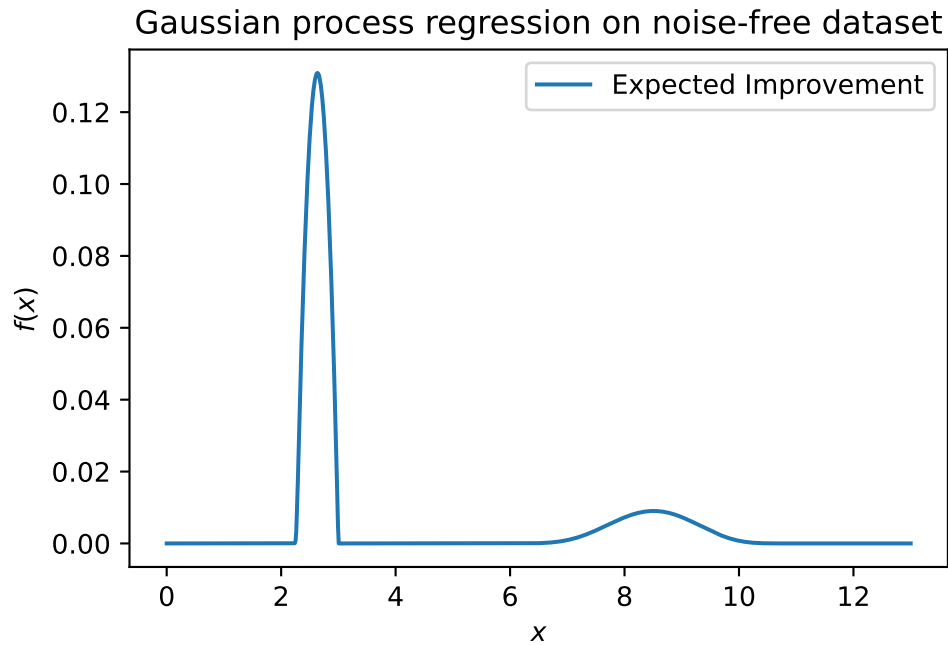
```



```
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)
```

```

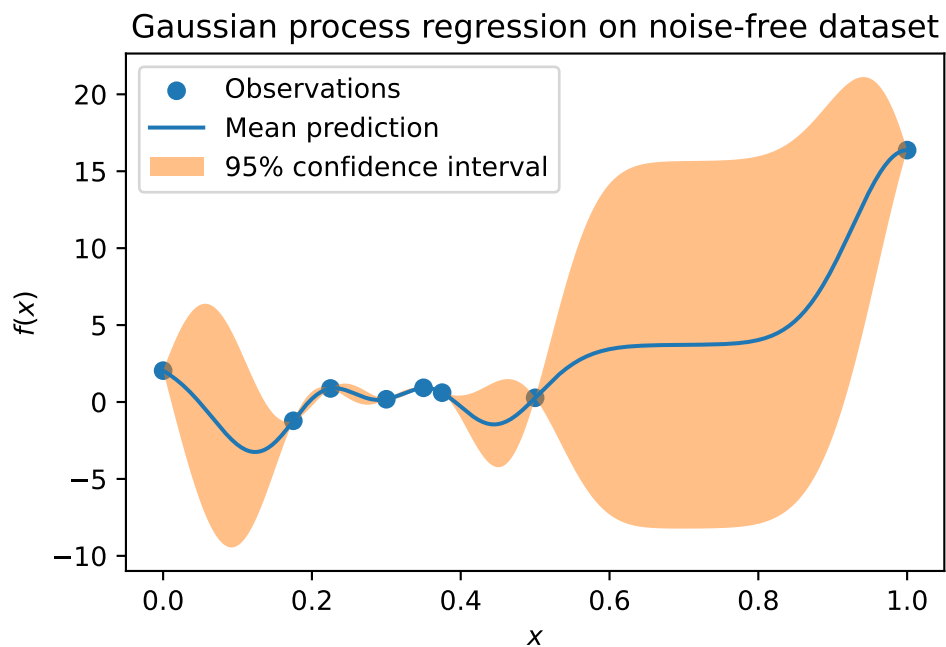
fun = analytical().fun_forrester
fun_control = {"sigma": 1.0,
               "seed": 123}
y_train = fun(X_train, fun_control=fun_control)

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")

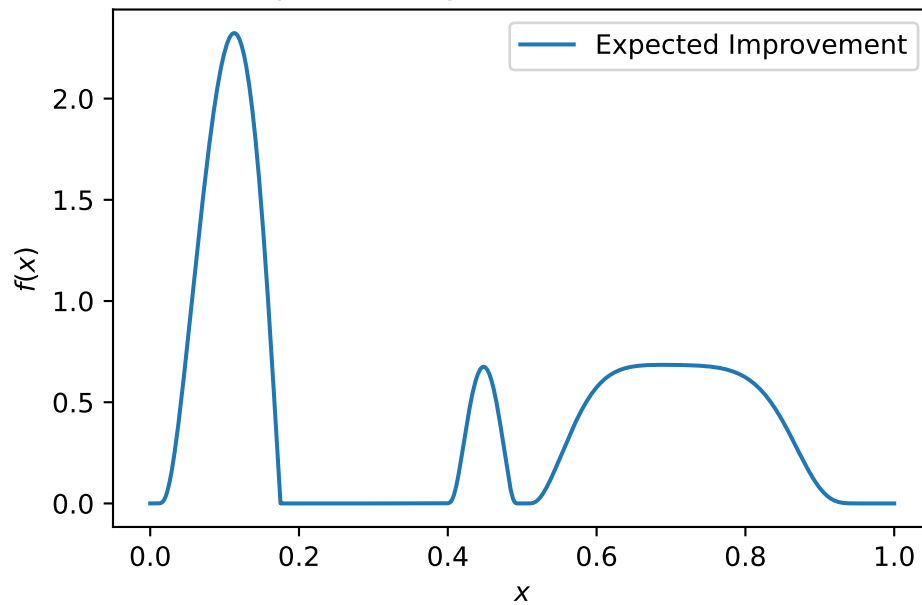
plt.scatter(X_train, y_train, label="Observations")
plt.plot(X, mean_prediction, label="Mean prediction")
if True:
    plt.fill_between(
        X.ravel(),
        mean_prediction - 2 * std_prediction,
        mean_prediction + 2 * std_prediction,
        alpha=0.5,
        label=r"95% confidence interval",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

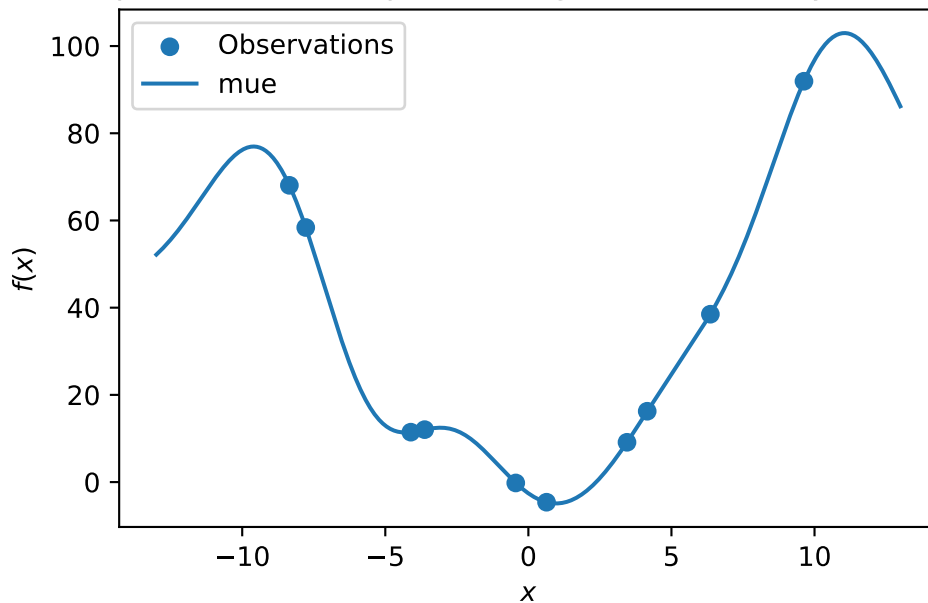
```

```

[[ 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)}
```

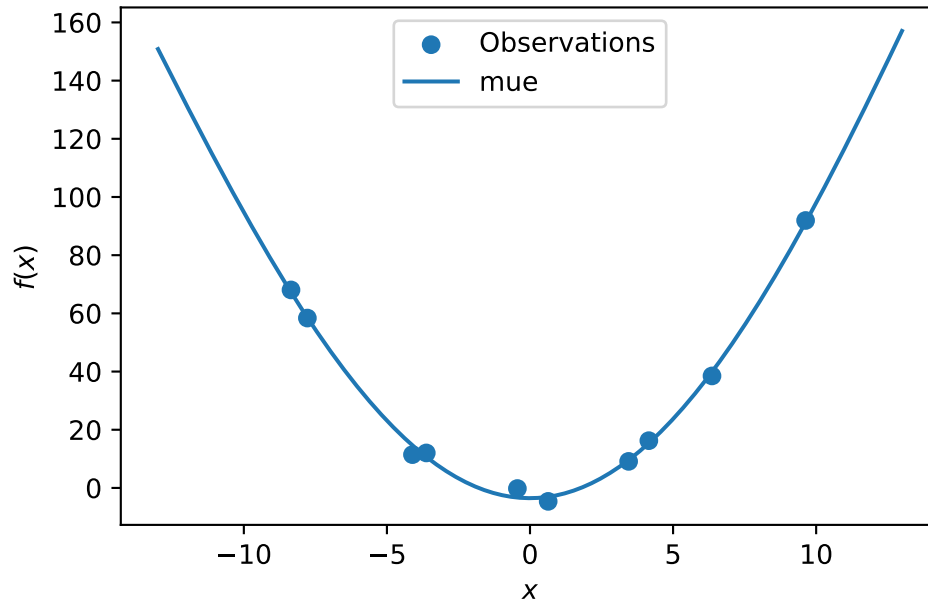
```
S = Kriging(name='kriging',
            seed=123,
            log_level=50,
            n_theta=1,
            noise=True)
S.fit(X_train, y_train)
```

```
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")

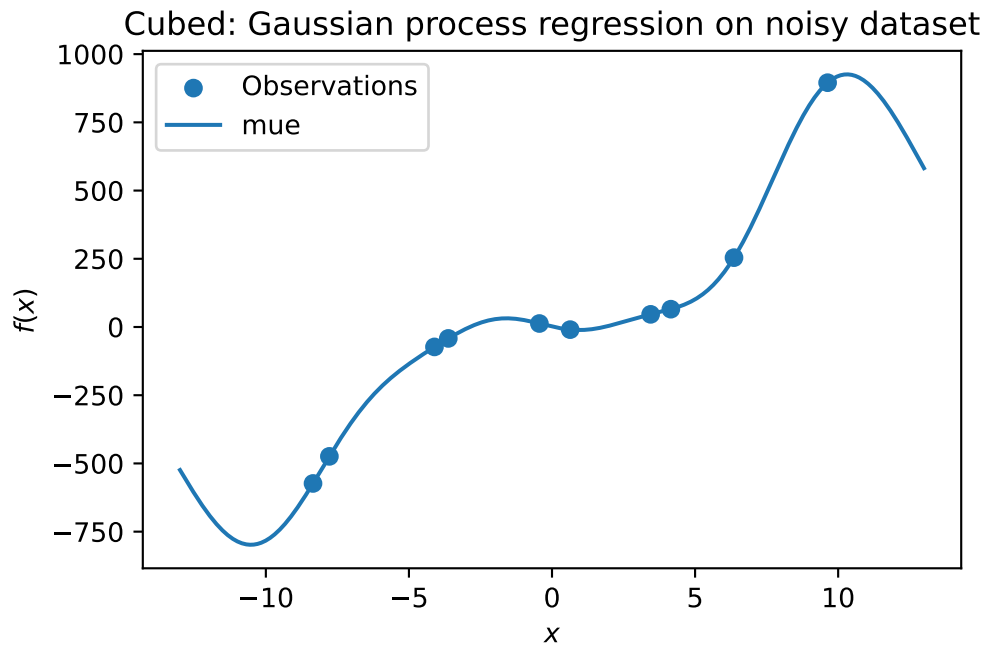
```

```

[[ 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]
```

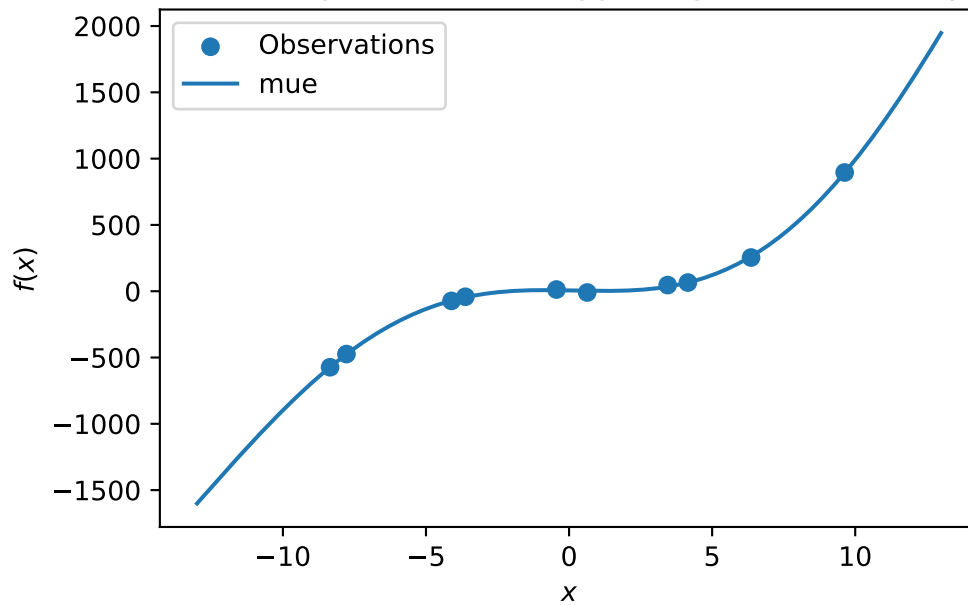


```
S = Kriging(name='kriging', seed=123, log_level=0, n_theta=1, noise=True)
S.fit(X_train, y_train)

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")

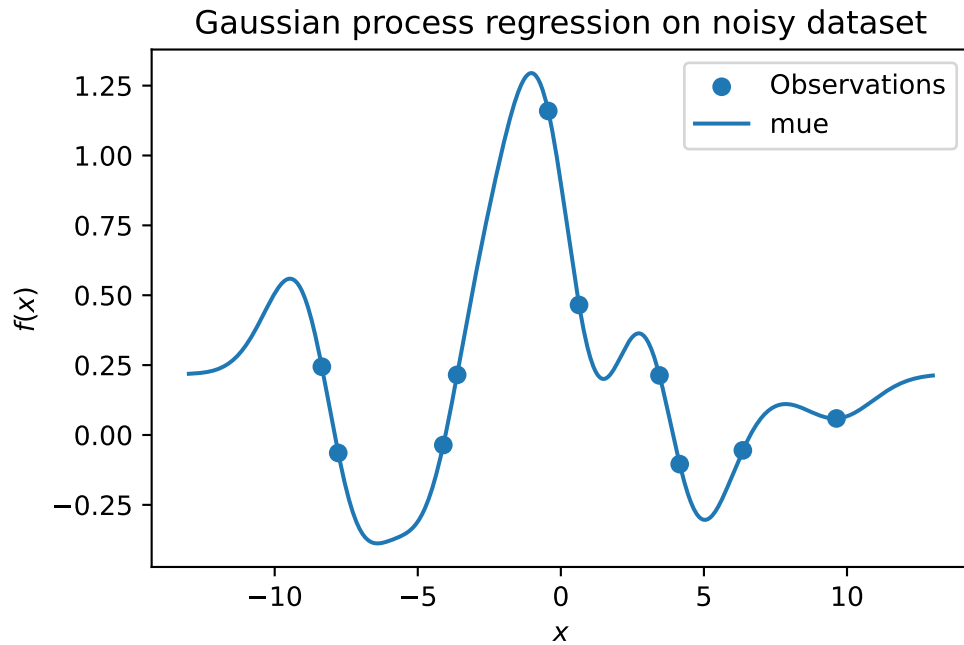
plt.scatter(X_train, y_train, label="Observations")
#plt.plot(X, ei, label="Expected Improvement")
plt.plot(X_axis, mean_prediction, label="mue")
plt.legend()
plt.xlabel("$x$")
plt.ylabel("$f(x)$")
_ = plt.title("Gaussian process regression on noisy dataset")

```

```

[[ 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]

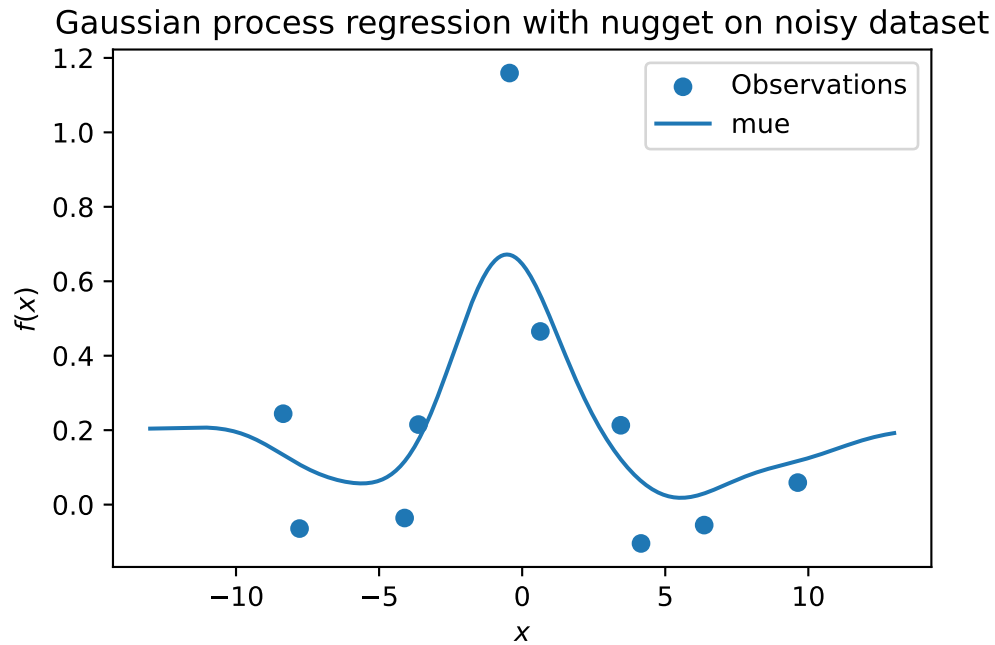
```



```
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

```

-212.49167147465846

```
# 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_2023-06-28_01-12-22

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:

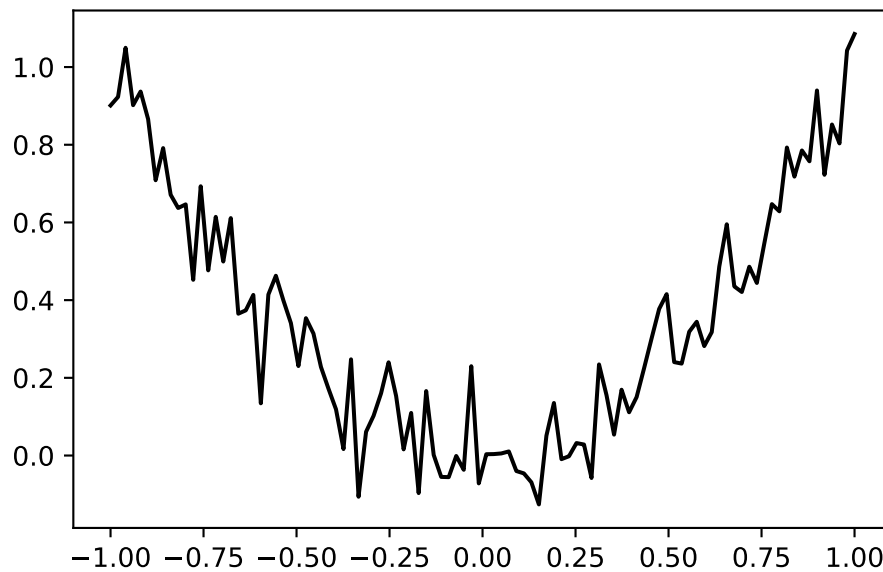
$$f(x) = x^2 + \epsilon$$

- 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()
```

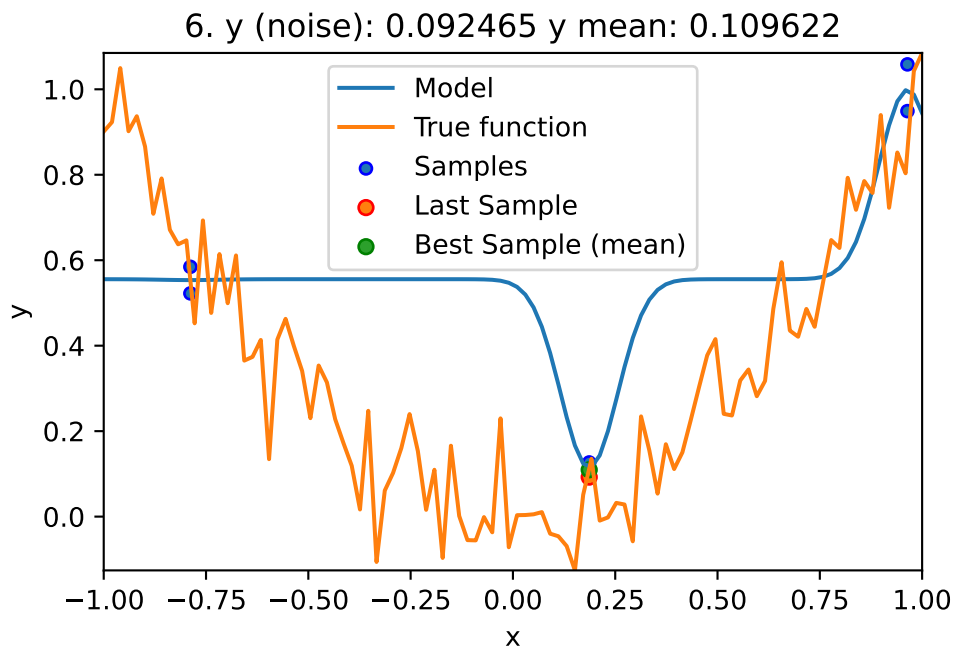


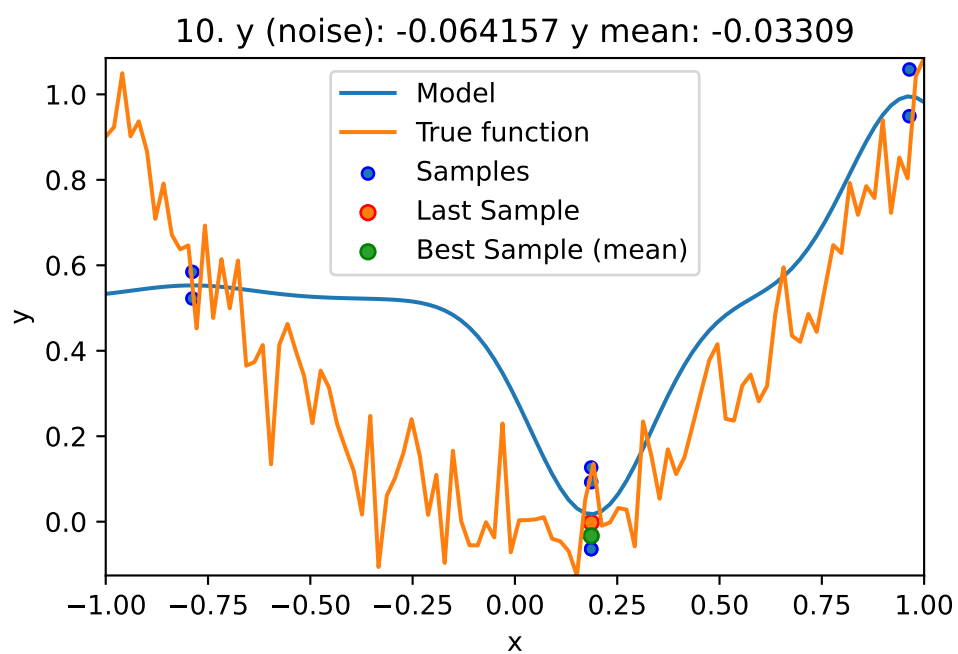
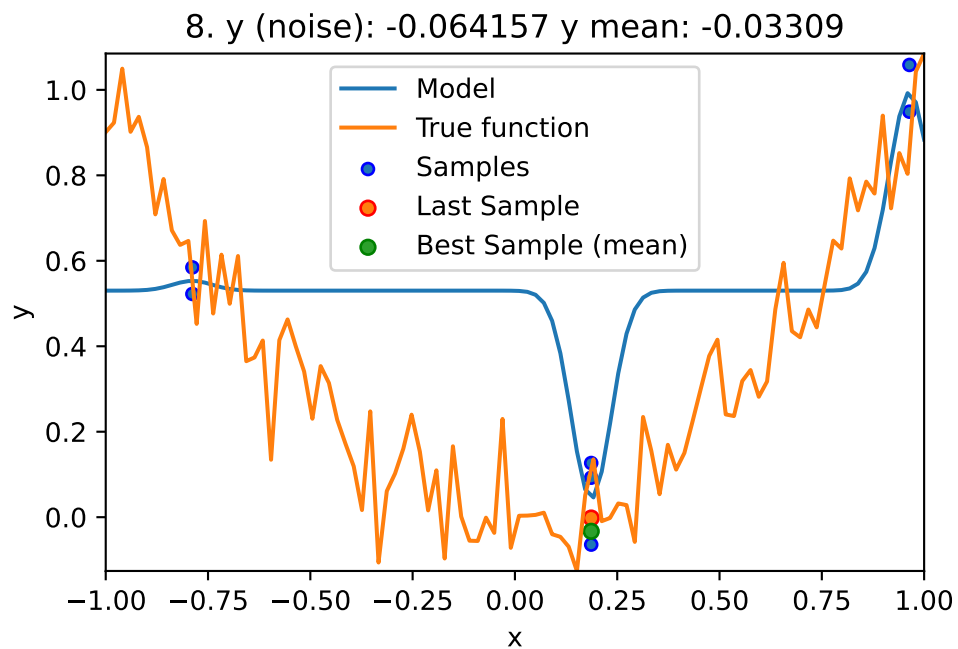
Spot is adopted as follows to cope with noisy functions:

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 0x15e1dcd00>

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")
```

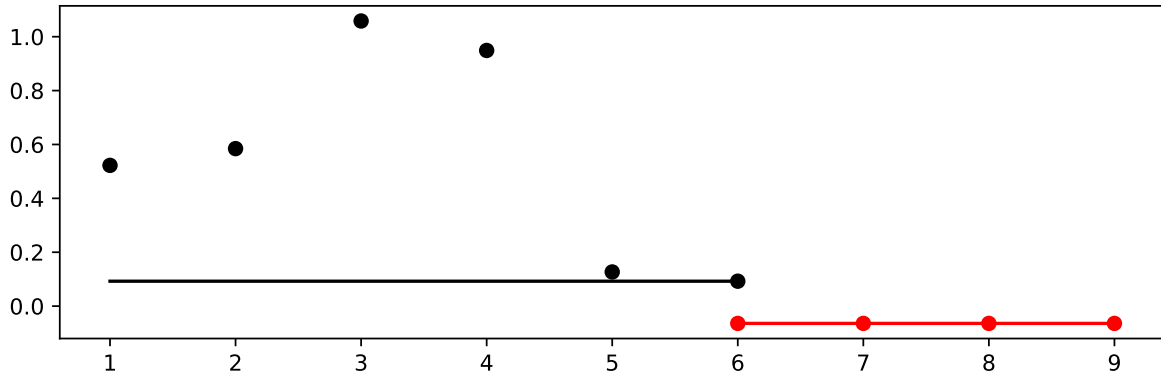


Figure 8.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

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:

```

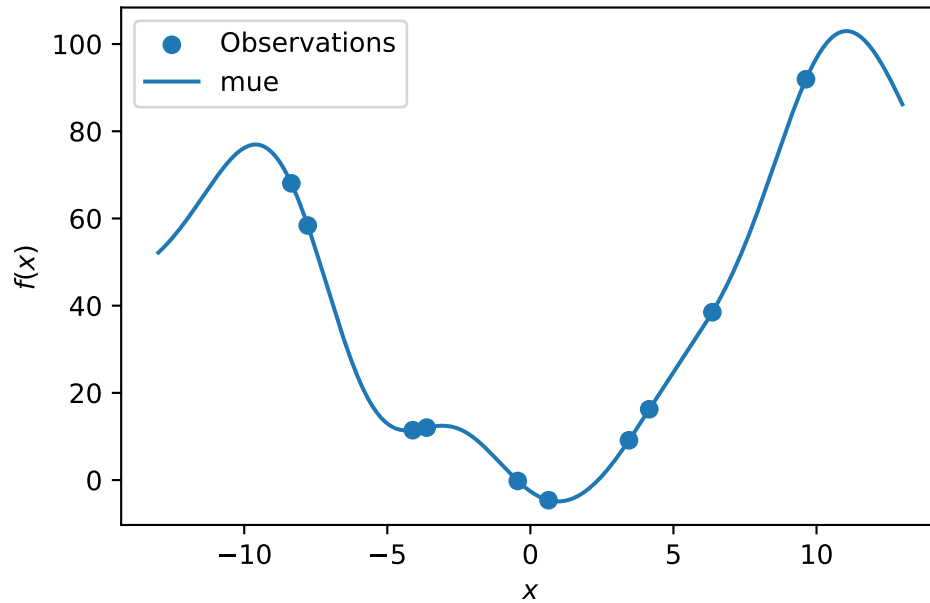
S = Kriging(name='kriging',
            seed=123,
            log_level=50,
            n_theta=1,
            noise=False)
S.fit(X_train, y_train)

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")

```

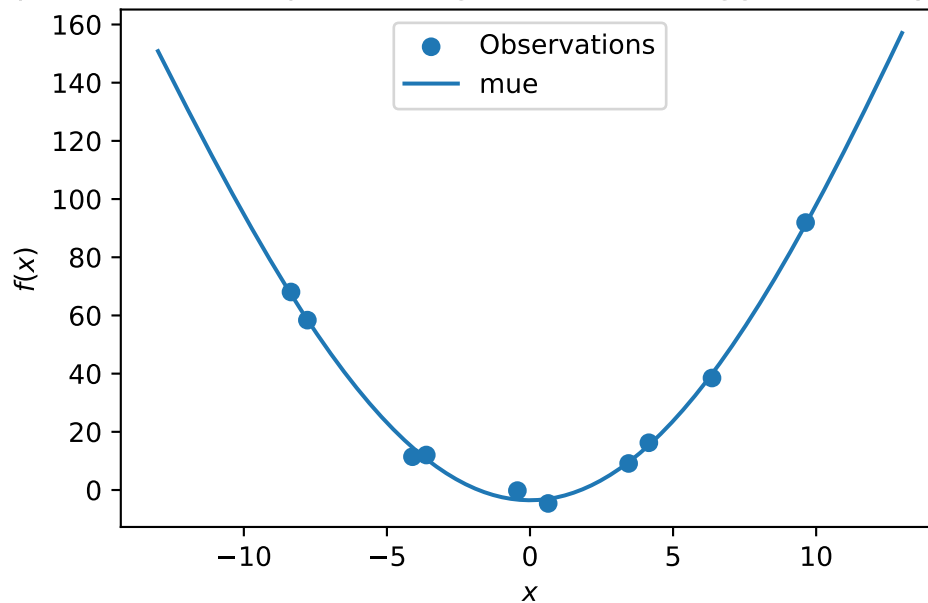
Sphere: Gaussian process regression on noisy dataset



- 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
```

```
9.088150096649695e-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:

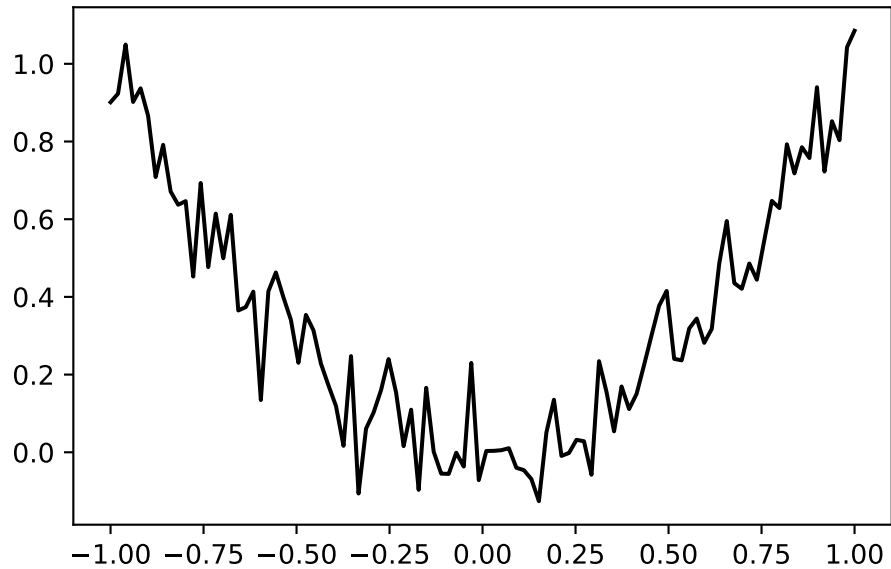
```
fun = analytical().fun_sphere
fun_control = {"sigma": 0.1,
              "seed": 123}
```

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,

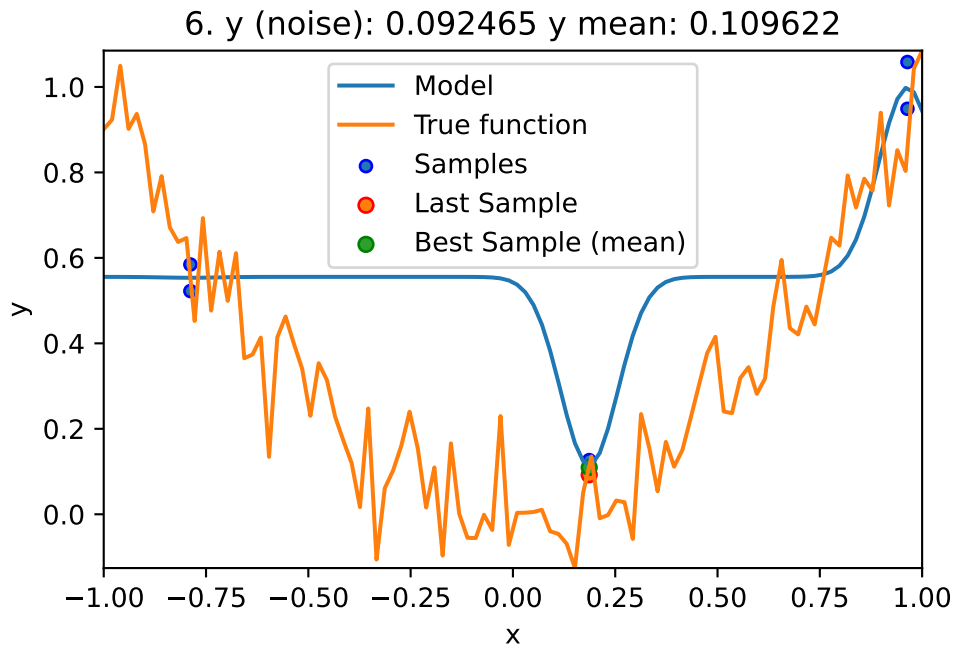
```

```

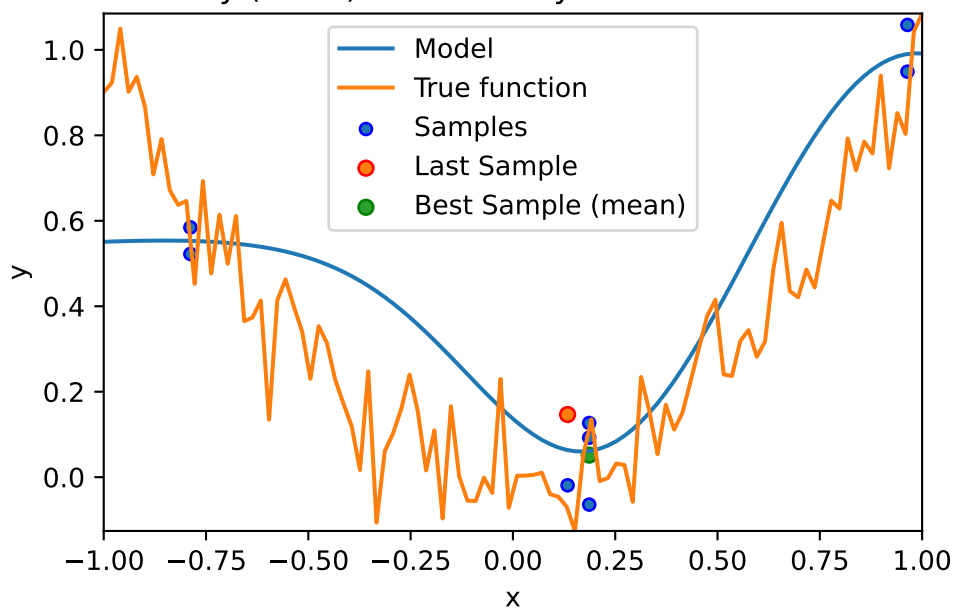
show_models=True,
fun_control = fun_control,
design_control={"init_size": 3,
               "repeats": 2},
surrogate_control={"noise": True})

```

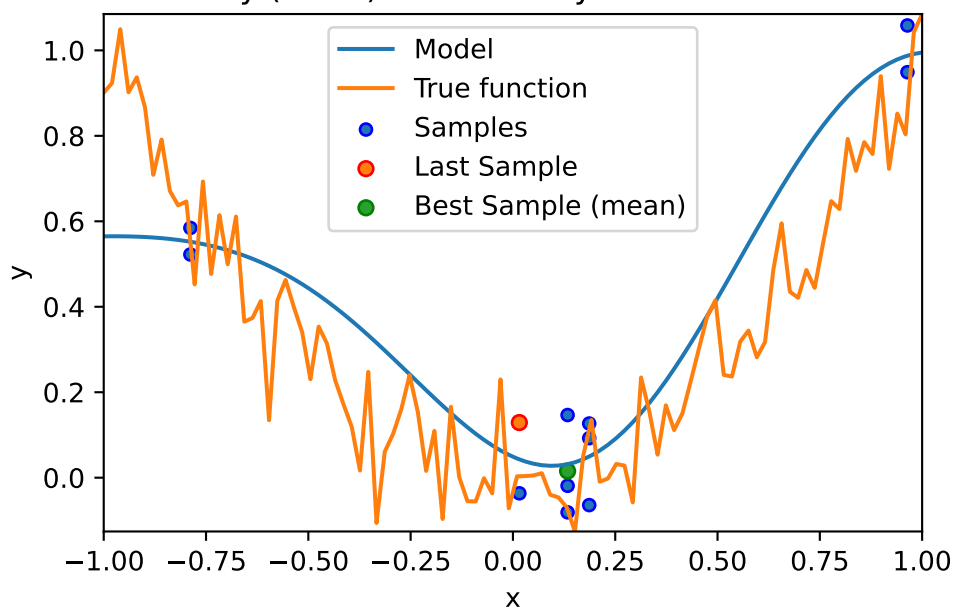
```
spot_1_noisy.run()
```

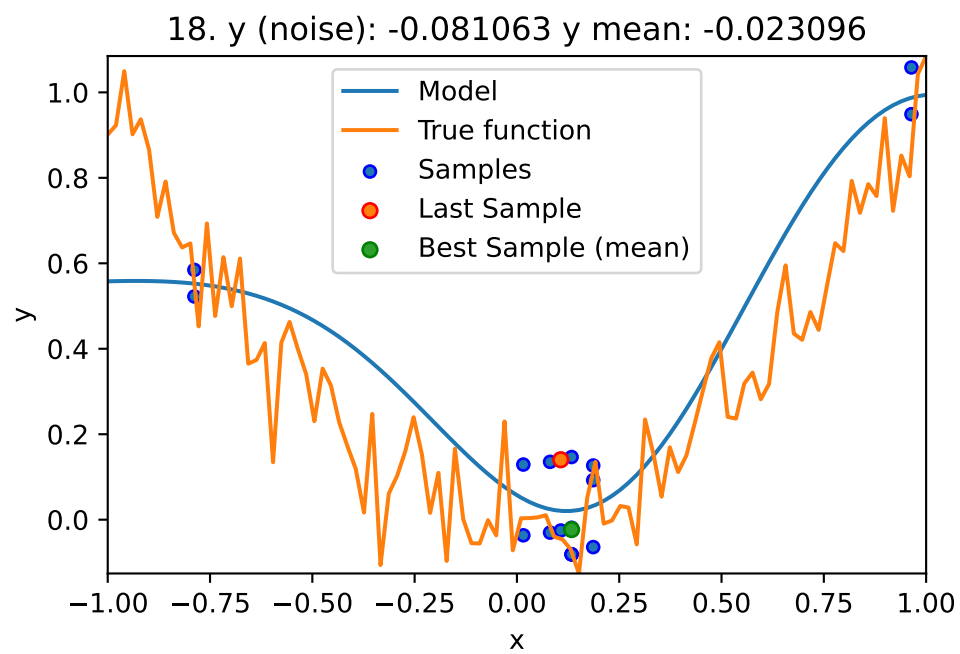
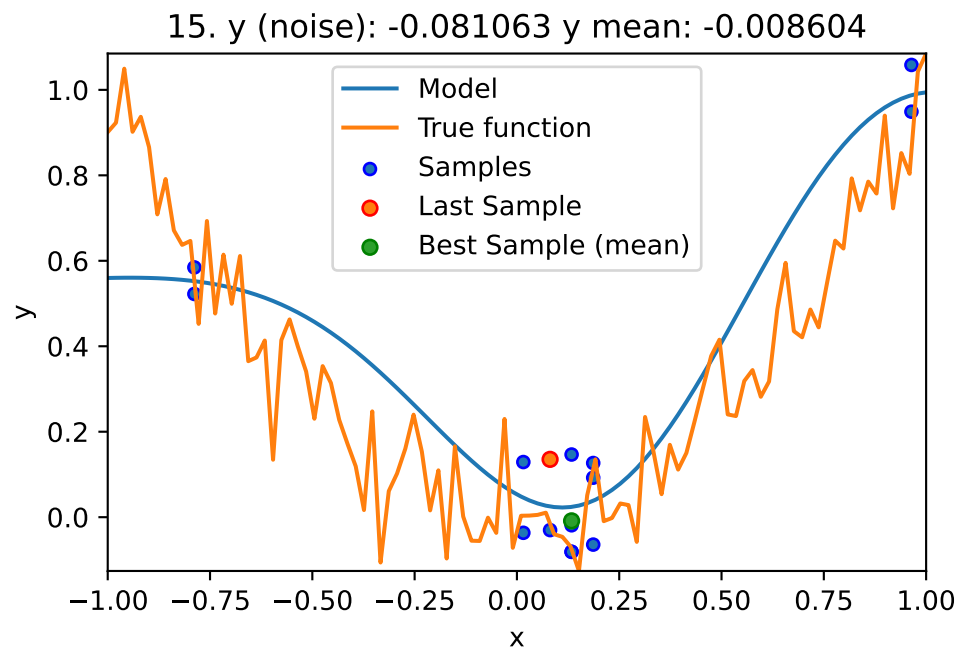


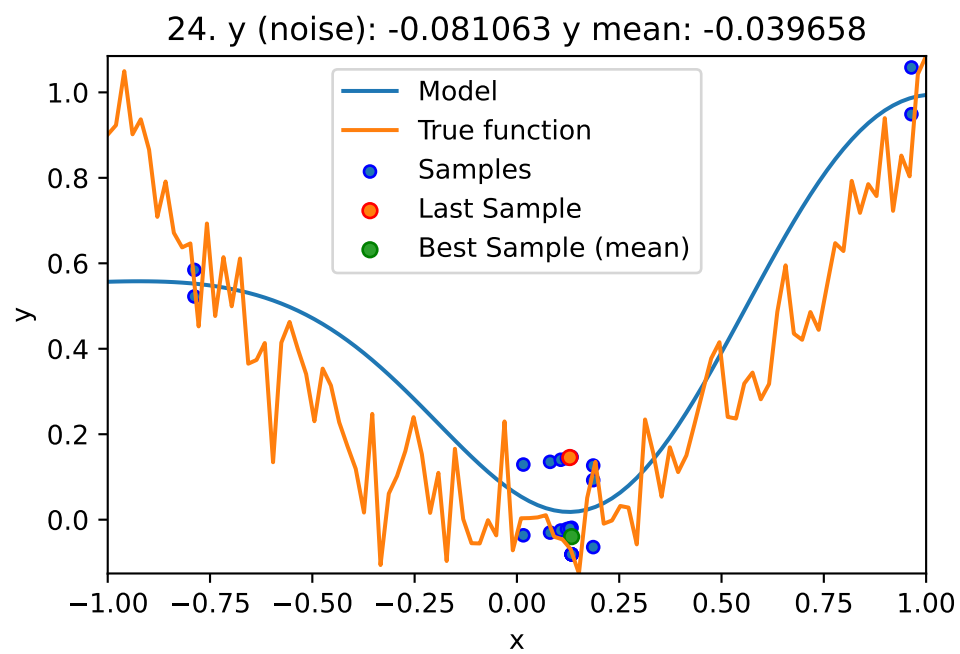
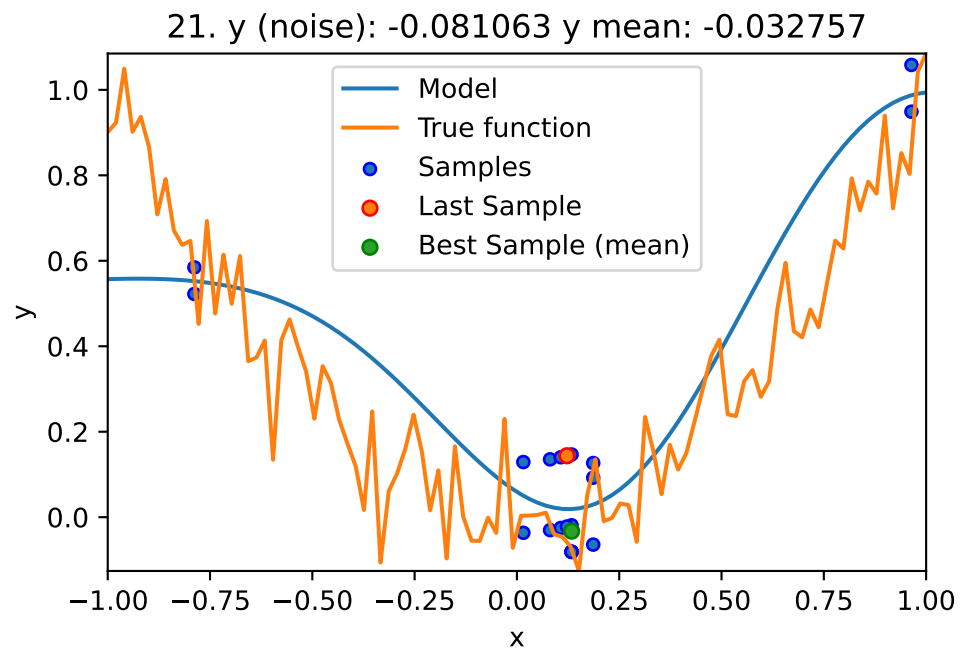
9. y (noise): -0.064157 y mean: 0.051695

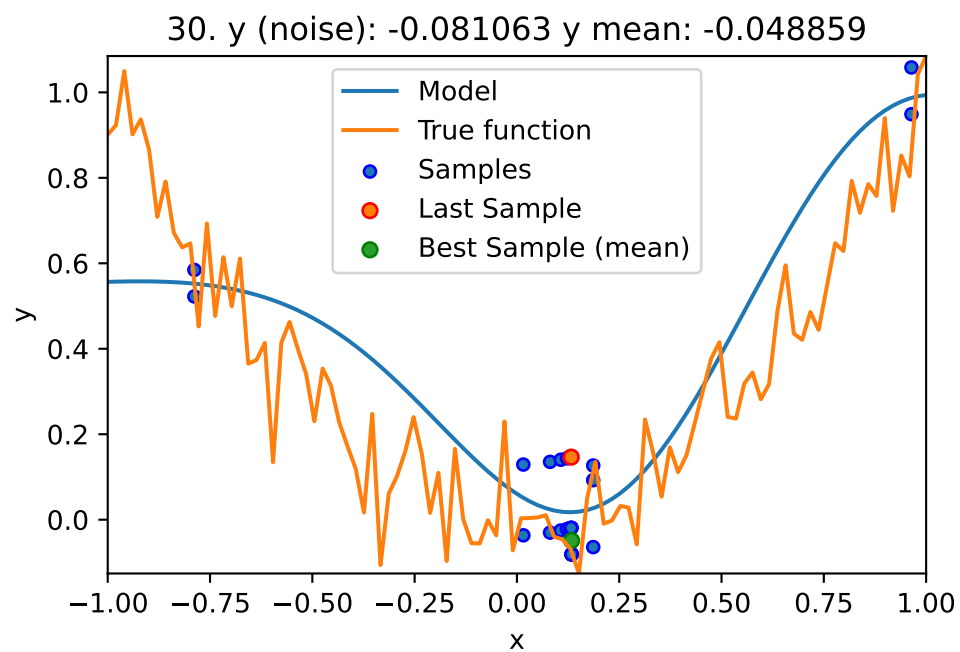
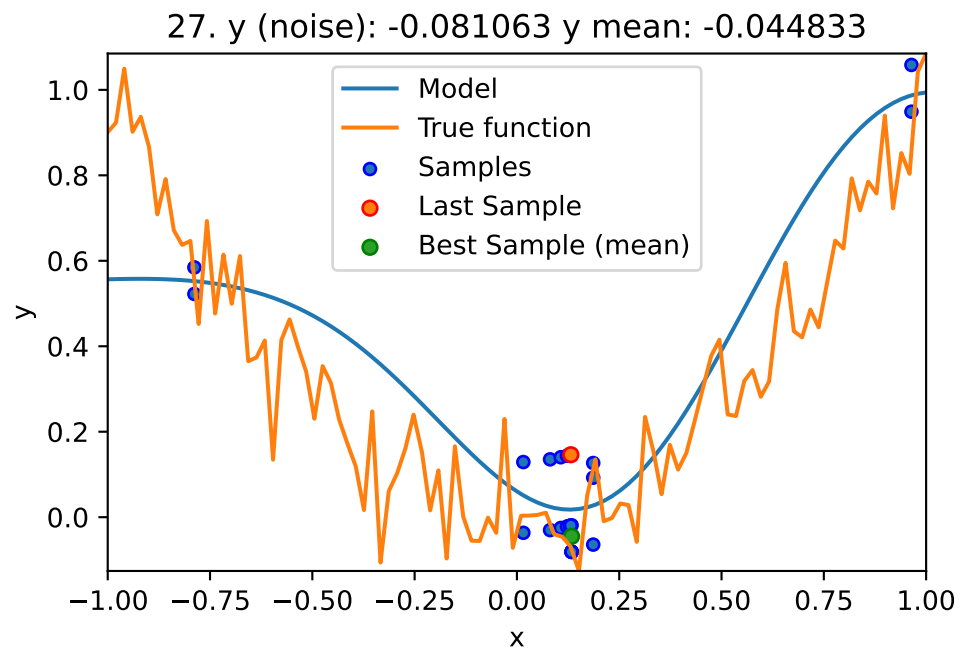


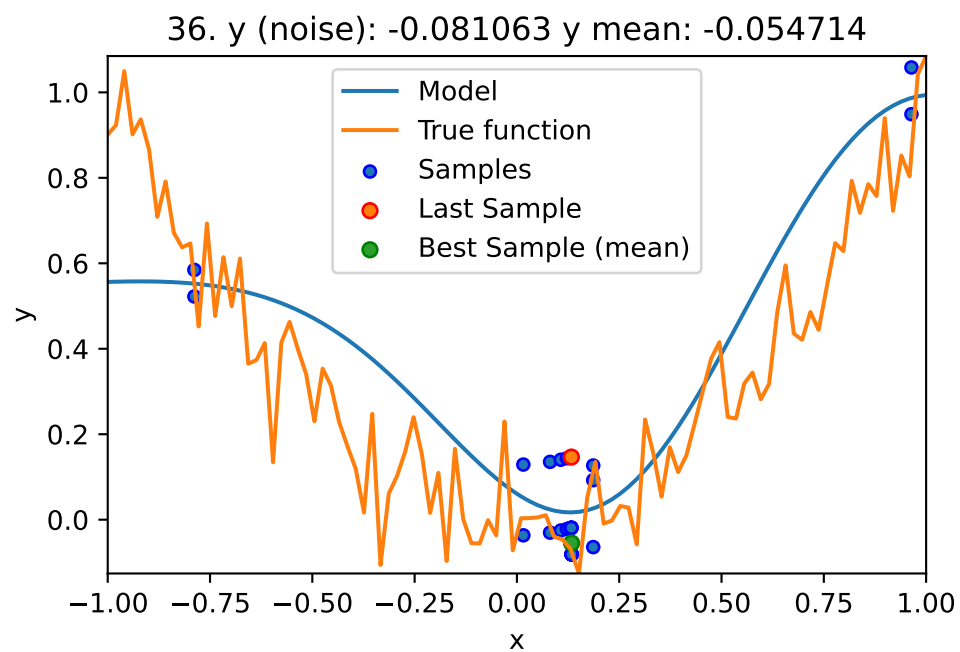
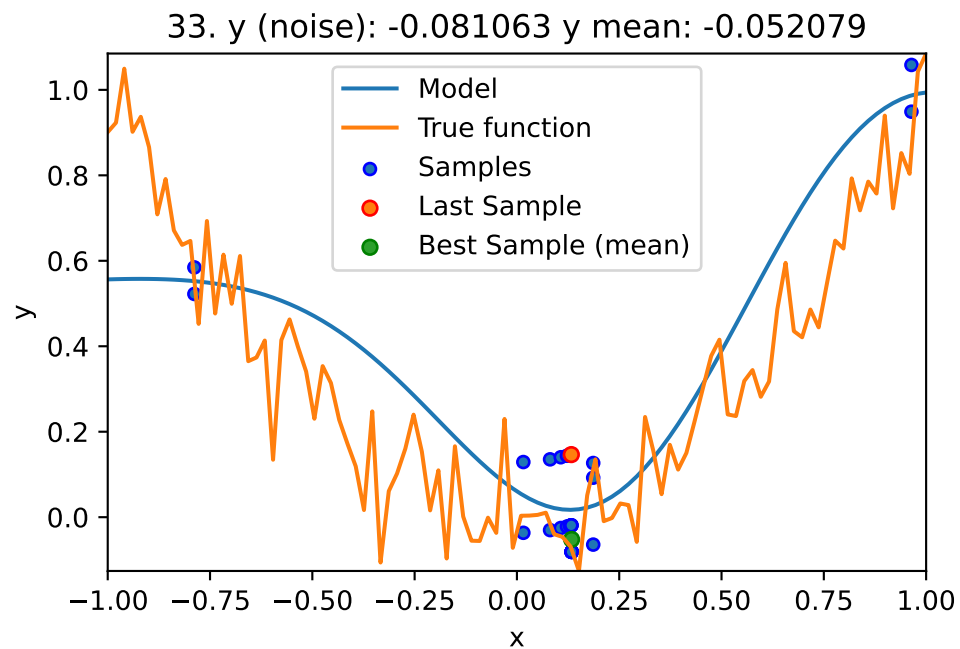
12. y (noise): -0.081063 y mean: 0.01555



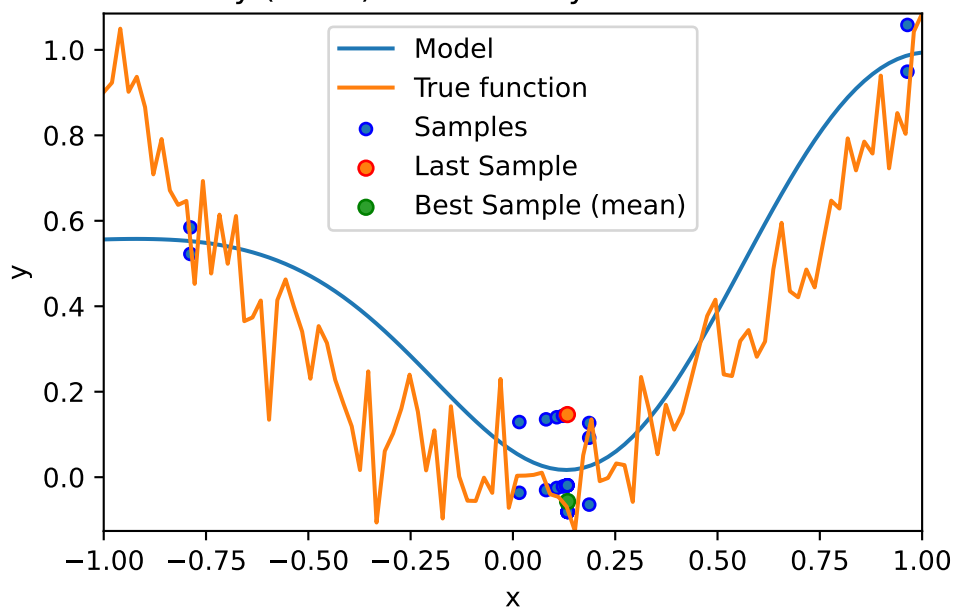




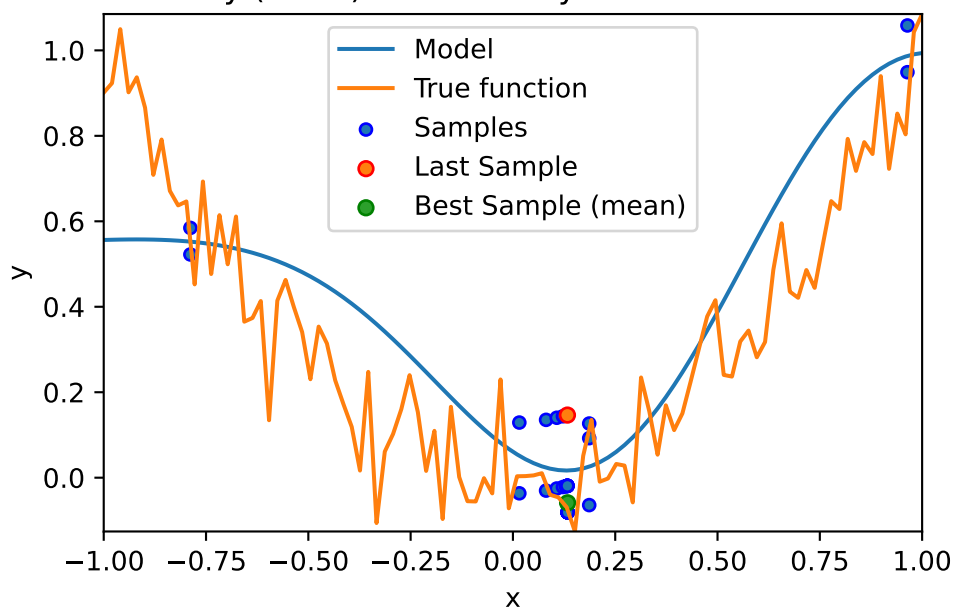




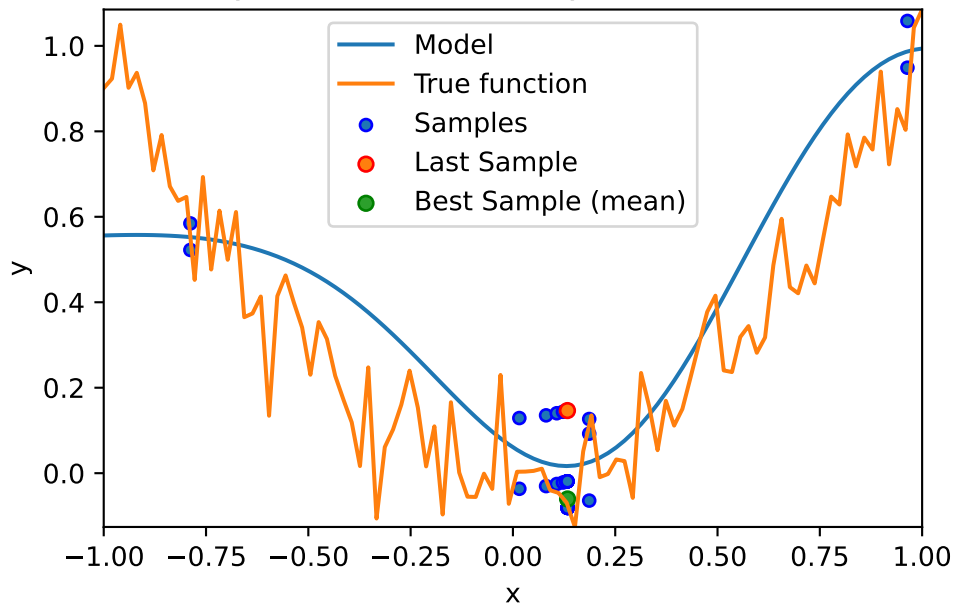
39. y (noise): -0.081063 y mean: -0.05691



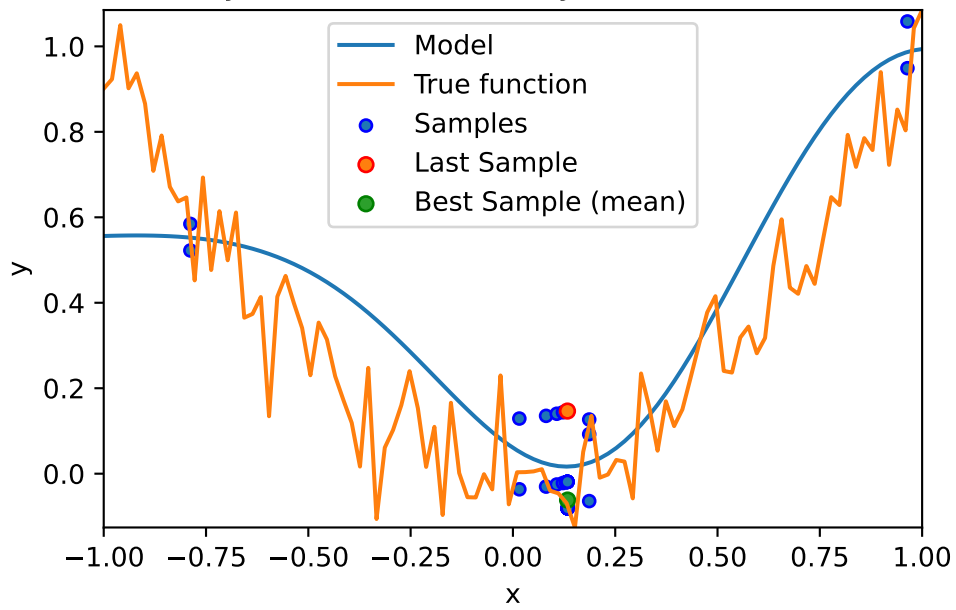
42. y (noise): -0.081063 y mean: -0.058768

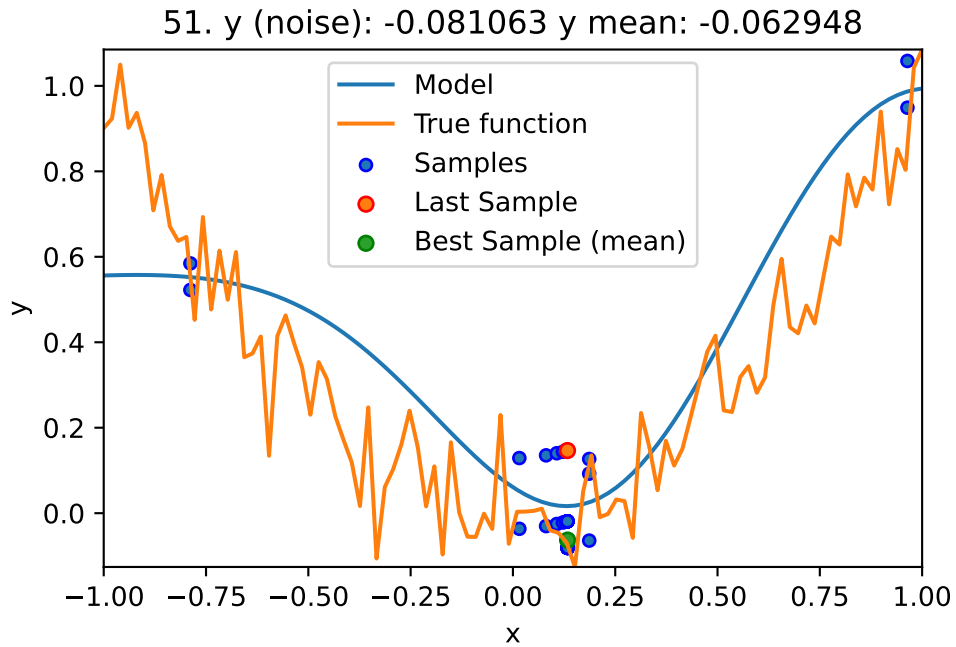


45. y (noise): -0.081063 y mean: -0.06036



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





```
<spotPython.spot.spot.Spot at 0x153b91390>
```

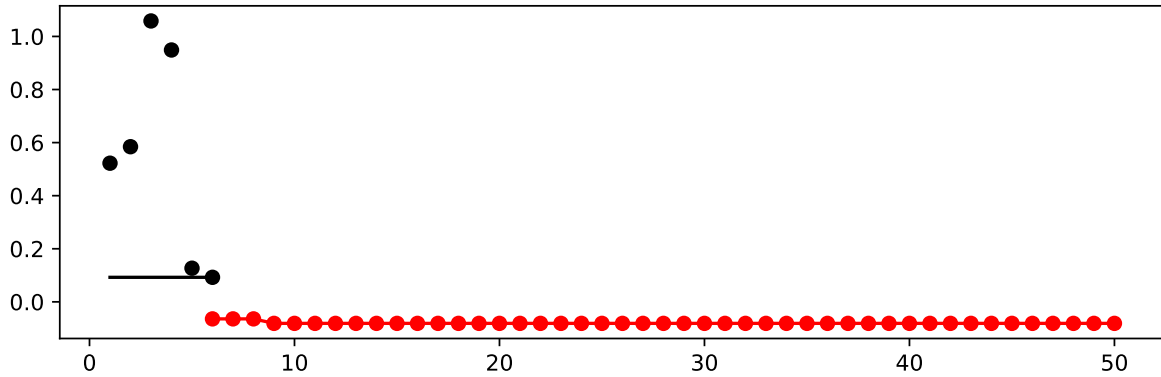
9.2 Print the Results

```
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:

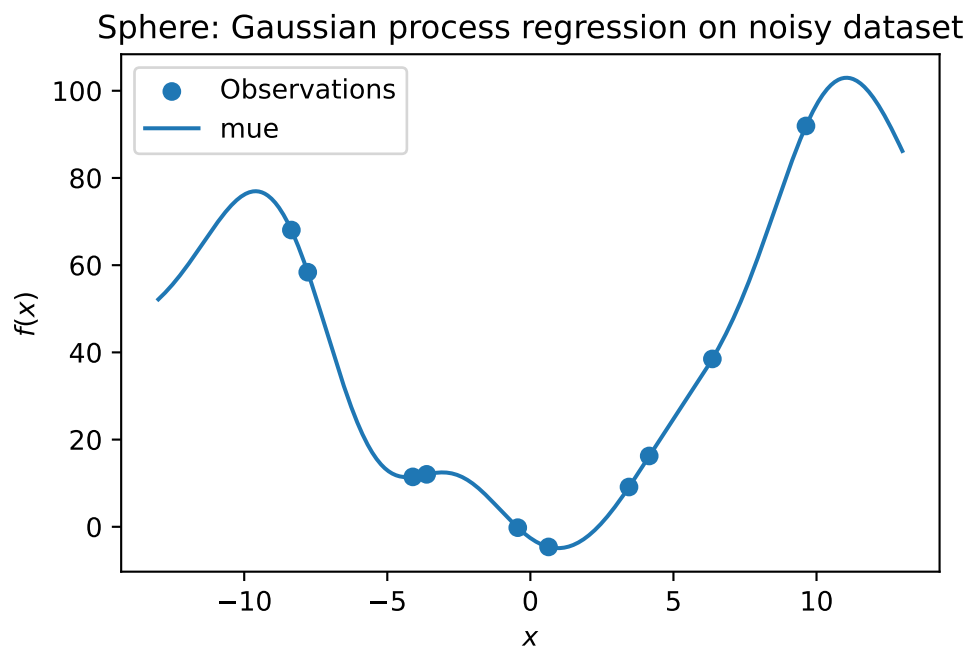
```

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")

```



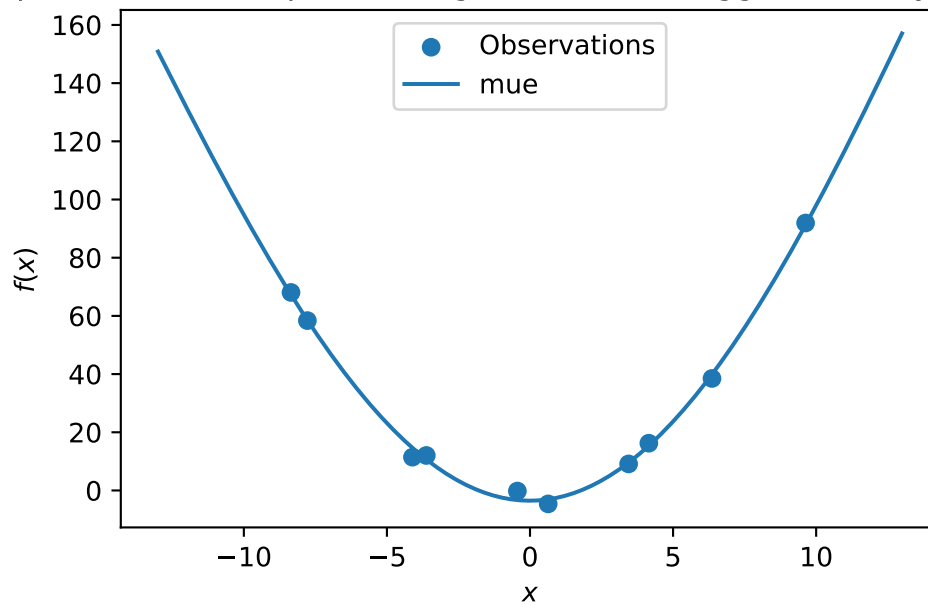
In comparison to the surrogate without nugget, we fit a surrogate with nugget to the data:

```

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
```

9.088150104540512e-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	0.2.50
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_2023-06-28_01-14-25

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

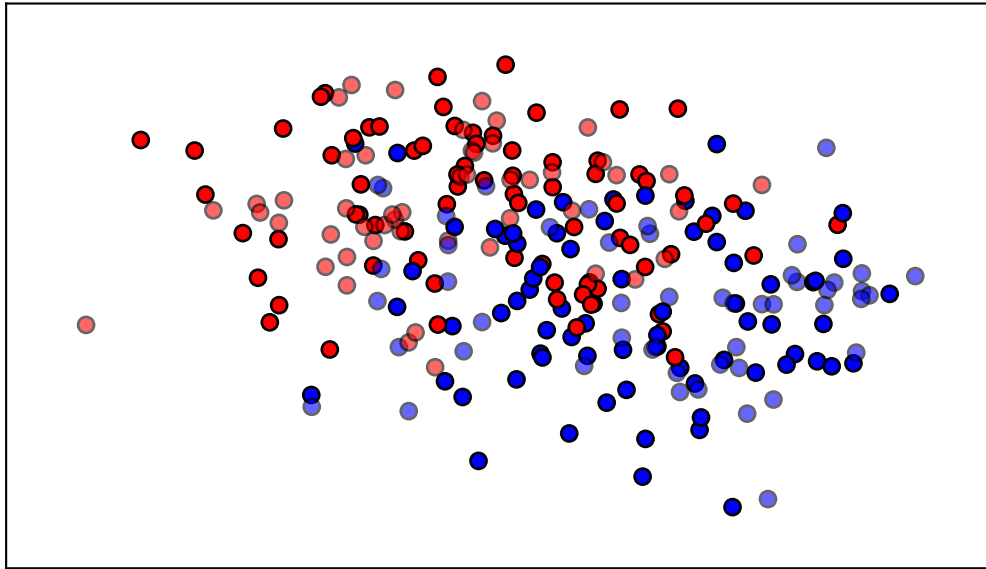
```

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.hypersklearn 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 [#-----] 7.27%

spotPython tuning: 4.7425859722522565 [#-----] 10.93%

spotPython tuning: 4.7425859722522565 [#-----] 14.38%

spotPython tuning: 4.7425859722522565 [##-----] 17.72%

```

spotPython tuning: 4.7425859722522565 [##-----] 20.73%

spotPython tuning: 4.7425859722522565 [##-----] 23.28%

spotPython tuning: 4.7425859722522565 [###-----] 32.63%

spotPython tuning: 4.7425859722522565 [####-----] 47.57%

spotPython tuning: 4.7425859722522565 [####-----] 51.37%

spotPython tuning: 4.7425859722522565 [####-----] 54.44%

spotPython tuning: 4.7425859722522565 [#####----] 70.34%

spotPython tuning: 4.7425859722522565 [#####----] 90.42%

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

<spotPython.spot.spot.Spot at 0x18514e860>

```

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")
```

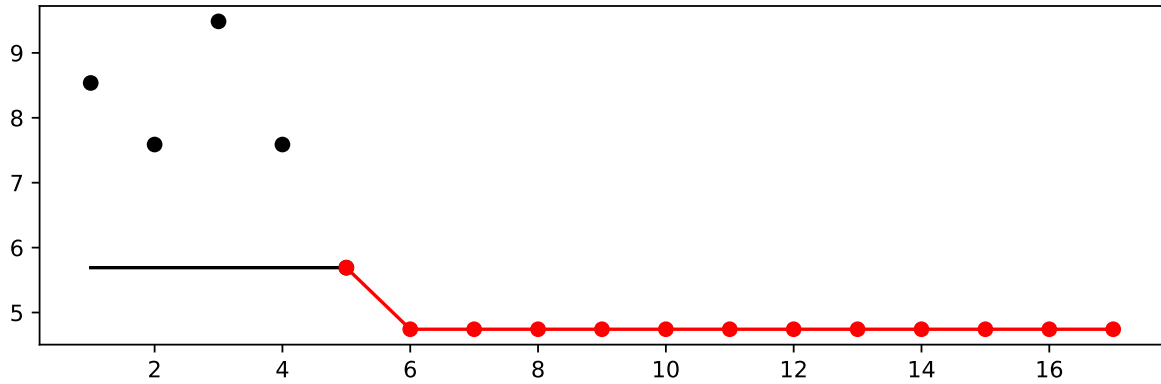


Figure 10.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- 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
```

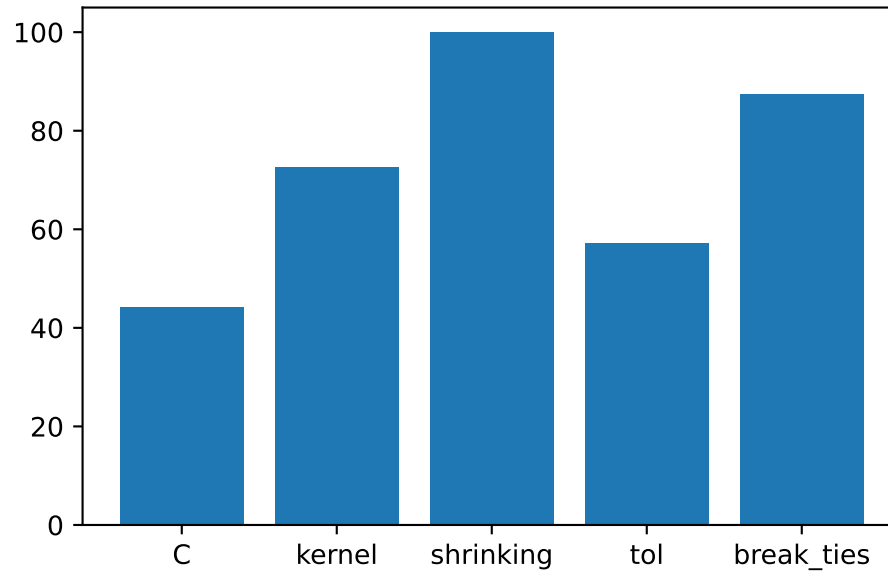


Figure 10.2: Variable importance plot, threshold 0.025.

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}]
```

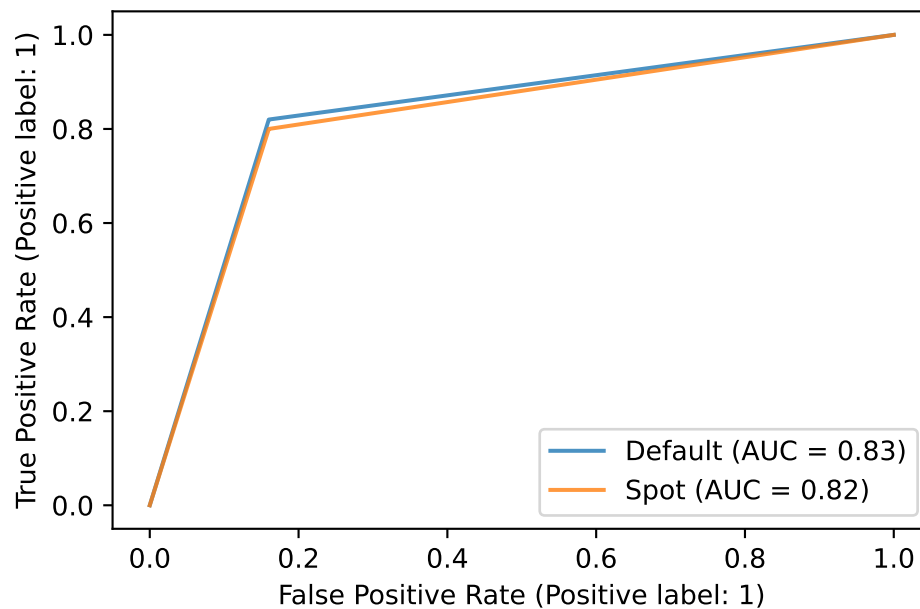
```
from spotPython.hyperparameters.values import get_one_sklearn_model_from_X
model_spot = get_one_sklearn_model_from_X(X, fun_control)
```

```
model_spot
```

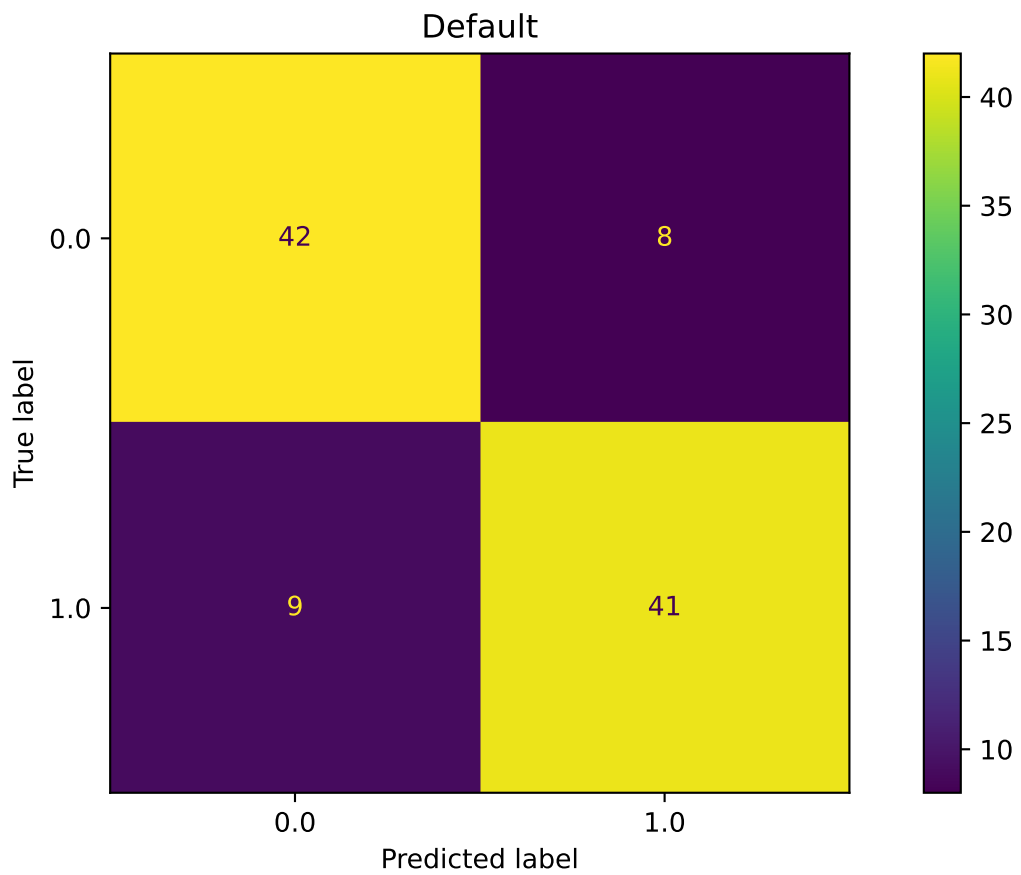
```
Pipeline(steps=[('standardscaler', StandardScaler()),  
                ('svc',  
                 SVC(C=0.23258412447782734, break_ties=1,  
                     cache_size=214.29269330654913, kernel='poly',  
                     probability=1, shrinking=1, tol=0.003757085413122674))])
```

10.9.4 Plot: Compare Predictions

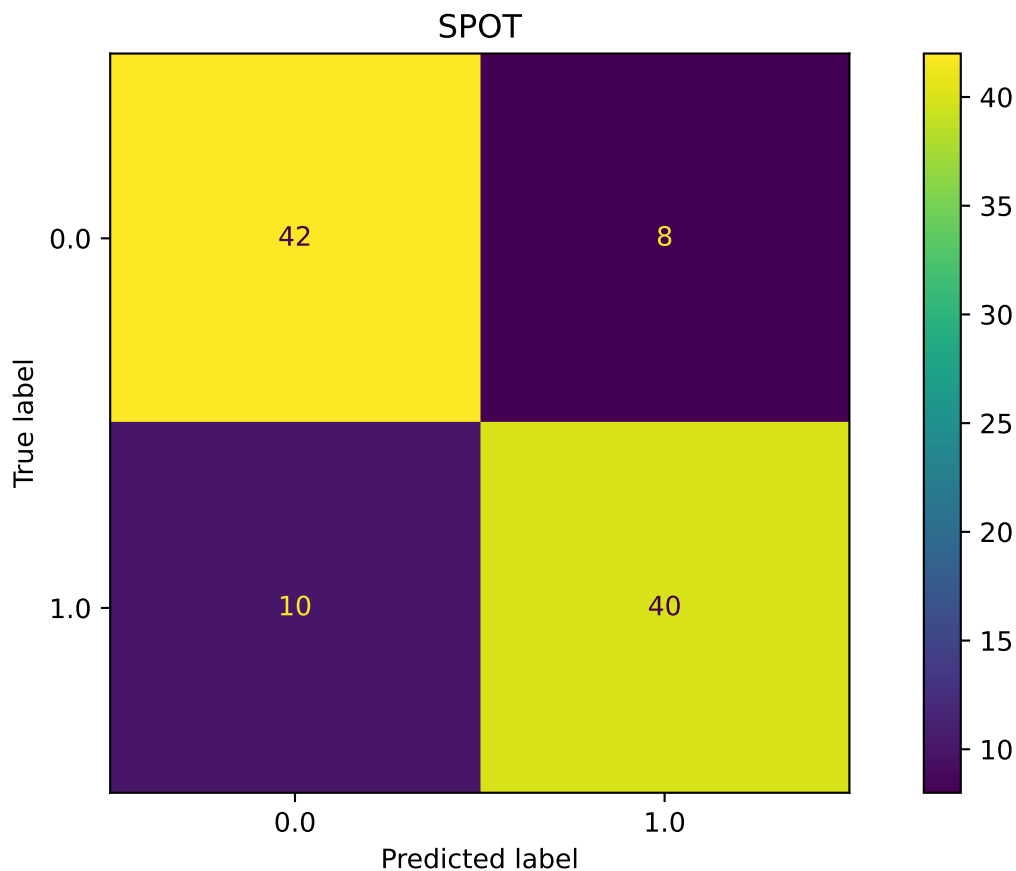
```
from spotPython.plot.validation import plot_roc  
plot_roc([model_default, model_spot], fun_control, model_names=["Default", "Spot"])
```



```
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")
```



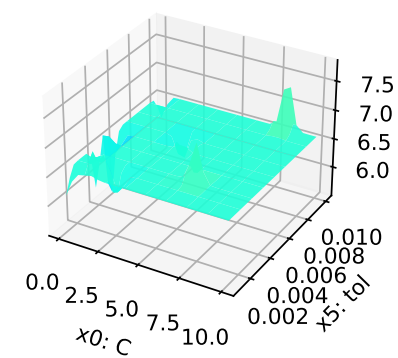
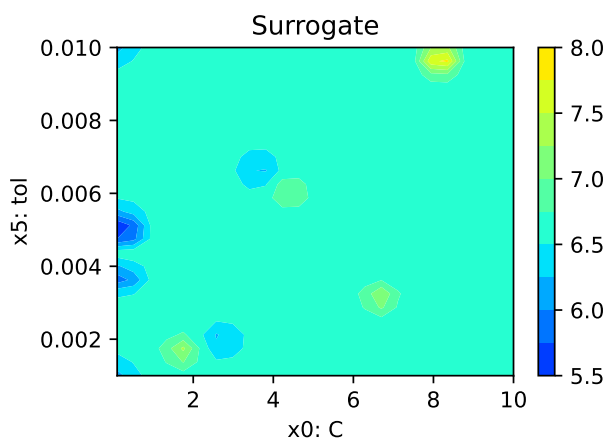
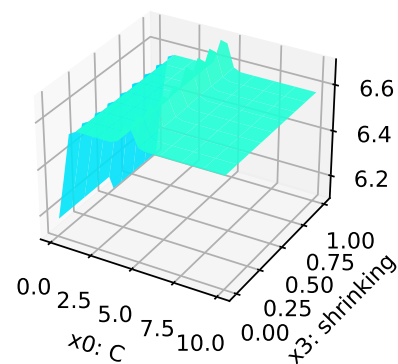
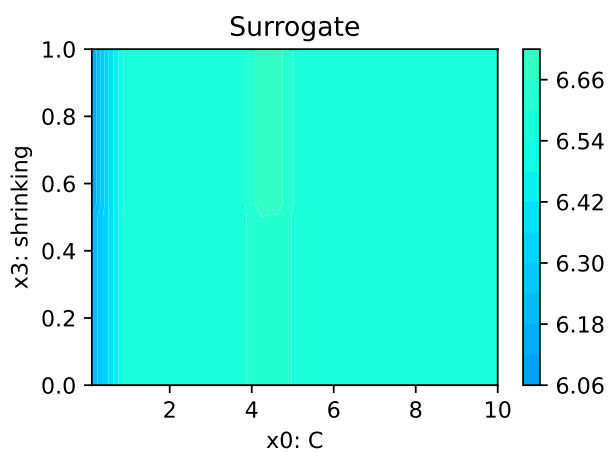
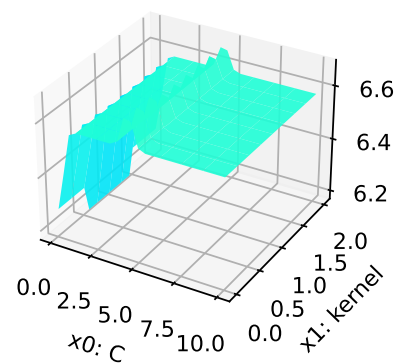
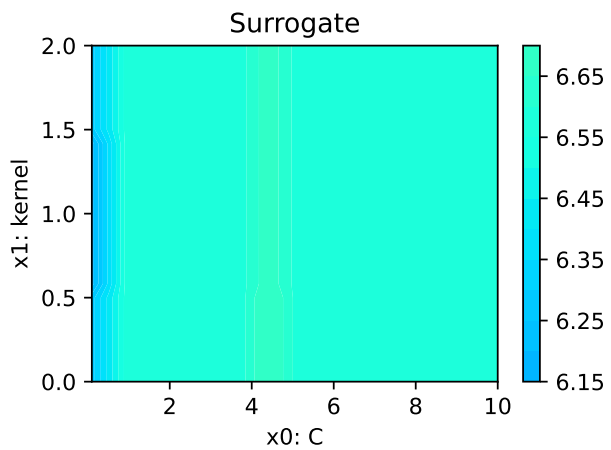
```
min(spot_tuner.y), max(spot_tuner.y)
```

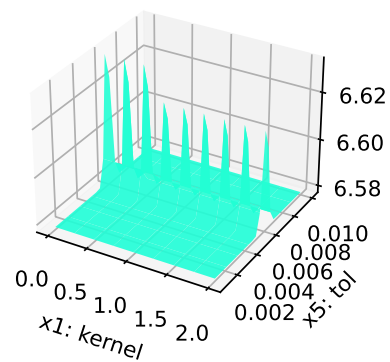
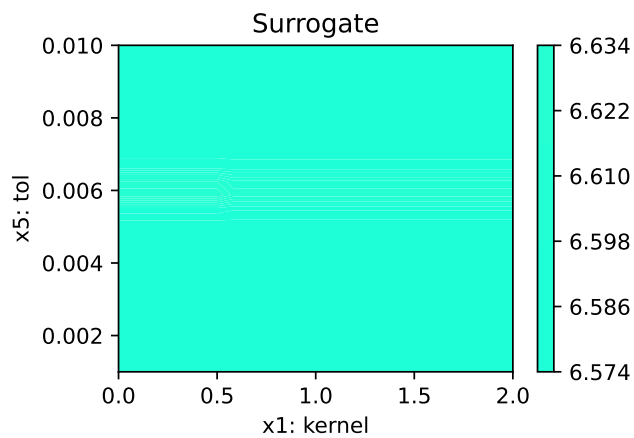
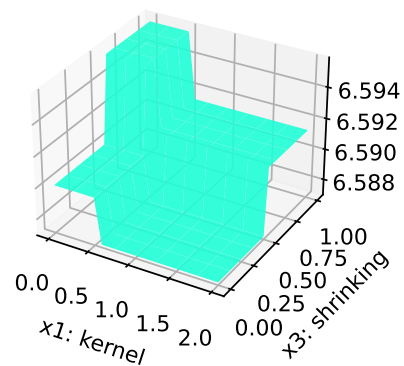
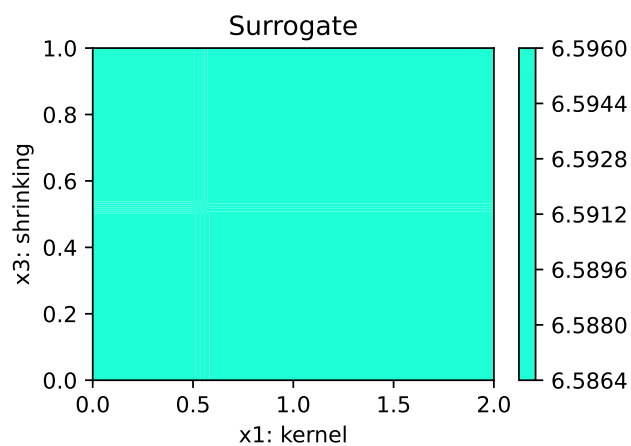
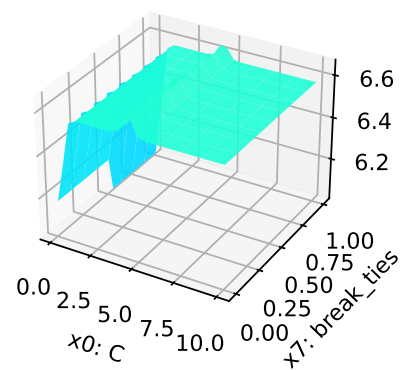
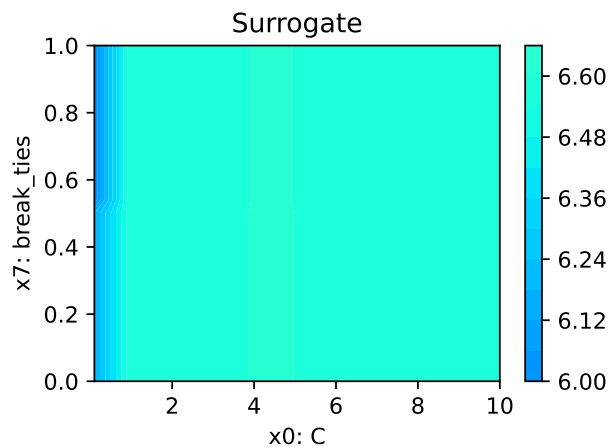
```
(4.7425859722522565, 9.485171944504513)
```

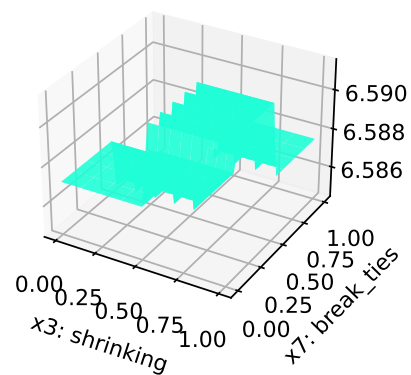
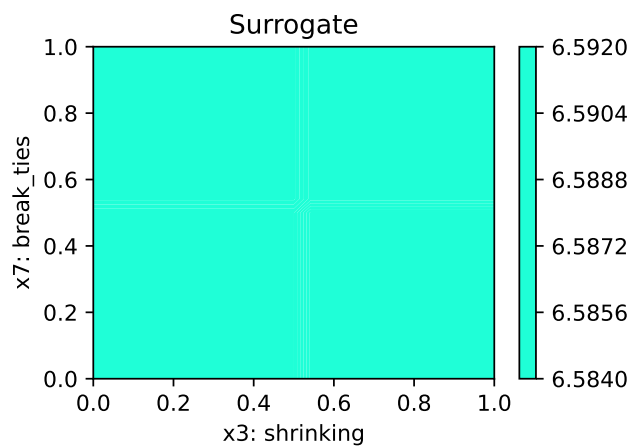
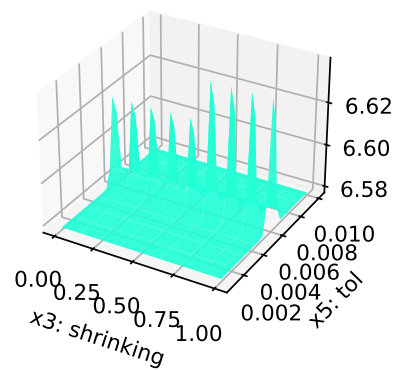
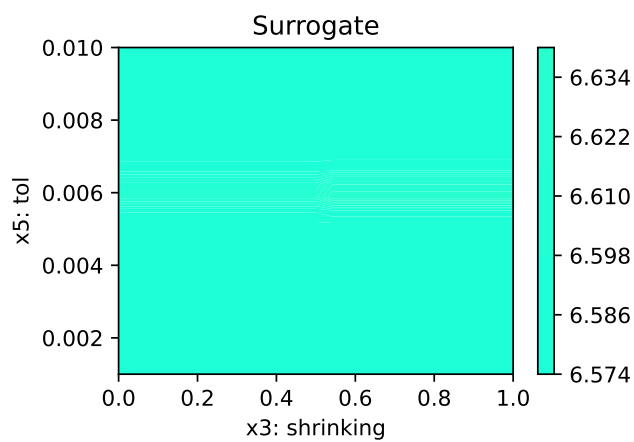
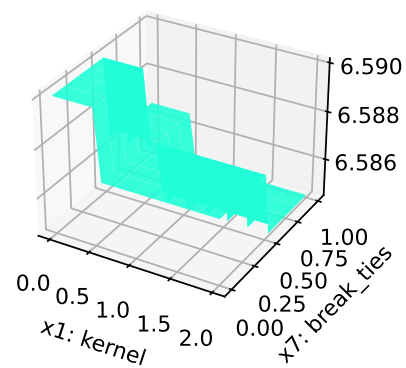
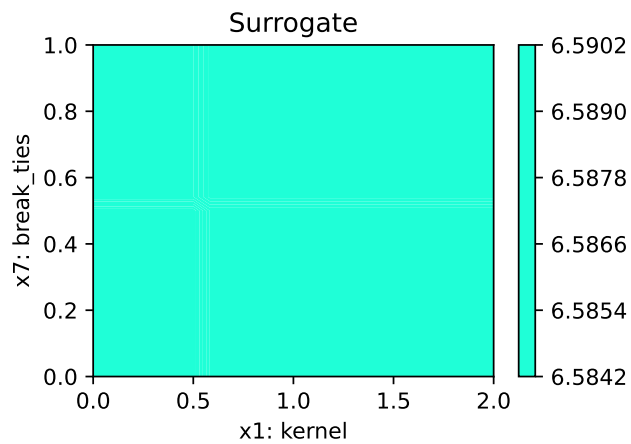
10.9.5 Detailed Hyperparameter Plots

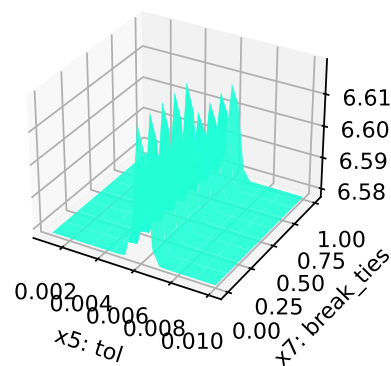
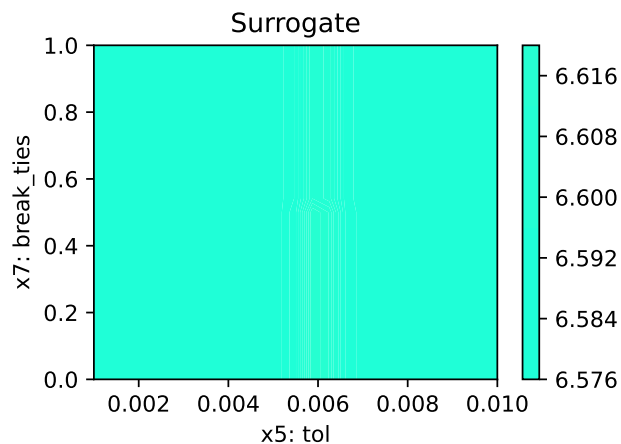
```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

```
C: 44.23043862790405
kernel: 72.50555365607846
shrinking: 99.99999999999999
tol: 57.11958727721194
break_ties: 87.43775176454929
```









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	0.2.50
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_2023-06-28_01-31-37

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: 0.3134583234786987 | Loss: 2.2125518328348797 | Acc: 0.3134583333333333.
Returned to Spot: Validation loss: 2.2125518328348797

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

MulticlassAccuracy: 0.2054583281278610 | Loss: 2.2771823295752207 | Acc: 0.2054583333333333.
Epoch: 2 |

MulticlassAccuracy: 0.2646666765213013 | Loss: 2.2488341910839083 | Acc: 0.2646666666666667.
Epoch: 3 |

MulticlassAccuracy: 0.2864583432674408 | Loss: 2.2168144498666127 | Acc: 0.2864583333333333.
Epoch: 4 |

MulticlassAccuracy: 0.2810416519641876 | Loss: 2.1846912537018457 | Acc: 0.2810416666666667.
Returned to Spot: Validation loss: 2.1846912537018457

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

MulticlassAccuracy: 0.3412916660308838 | Loss: 2.1538057389259340 | Acc: 0.3412916666666667.
Epoch: 2 |

MulticlassAccuracy: 0.3619166612625122 | Loss: 1.9509642014106114 | Acc: 0.3619166666666667.
Epoch: 3 |

MulticlassAccuracy: 0.4556666612625122 | Loss: 1.7532034409393866 | Acc: 0.4556666666666667.
Epoch: 4 |

MulticlassAccuracy: 0.4846250116825104 | Loss: 1.5527006690204144 | Acc: 0.4846250000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5006250143051147 | Loss: 1.3775845926503341 | Acc: 0.5006250000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5164583325386047 | Loss: 1.2329607445833584 | Acc: 0.5164583333333334.
Epoch: 7 |

MulticlassAccuracy: 0.5888333320617676 | Loss: 1.1181755936164408 | Acc: 0.5888333333333333.
Epoch: 8 |

MulticlassAccuracy: 0.6452083587646484 | Loss: 1.0303540887183820 | Acc: 0.6452083333333334.
Returned to Spot: Validation loss: 1.030354088718382

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

MulticlassAccuracy: 0.1589999943971634 | Loss: 2.3029267905155817 | Acc: 0.15900000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.1720000058412552 | Loss: 2.2830104693969089 | Acc: 0.17200000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.1730416715145111 | Loss: 2.2671343727310500 | Acc: 0.17304166666666667.
Epoch: 4 |

MulticlassAccuracy: 0.1727499961853027 | Loss: 2.2524220563769339 | Acc: 0.17275000000000000.
Returned to Spot: Validation loss: 2.252422056376934

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

MulticlassAccuracy: 0.2090833336114883 | Loss: 2.2665506587823230 | Acc: 0.20908333333333333.
Epoch: 2 |

MulticlassAccuracy: 0.2627499997615814 | Loss: 2.2297202168305716 | Acc: 0.26275000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.3014583289623260 | Loss: 2.1850140518347421 | Acc: 0.30145833333333333.
Epoch: 4 |

MulticlassAccuracy: 0.3260416686534882 | Loss: 2.1378745806217192 | Acc: 0.32604166666666667.
Epoch: 5 |

MulticlassAccuracy: 0.3449166715145111 | Loss: 2.0877542876005171 | Acc: 0.34491666666666666.
Epoch: 6 |

MulticlassAccuracy: 0.3699166774749756 | Loss: 2.0352002425591151 | Acc: 0.3699166666666667.
Epoch: 7 |

MulticlassAccuracy: 0.4181250035762787 | Loss: 1.9806690313816071 | Acc: 0.4181250000000000.
Epoch: 8 |

MulticlassAccuracy: 0.4687500000000000 | Loss: 1.9235149480899174 | Acc: 0.4687500000000000.
Returned to Spot: Validation loss: 1.9235149480899174

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

MulticlassAccuracy: 0.2295833379030228 | Loss: 2.2733383285999298 | Acc: 0.2295833333333333.
Epoch: 2 |

MulticlassAccuracy: 0.2315416634082794 | Loss: 2.2397762707869213 | Acc: 0.2315416666666667.
Epoch: 3 |

MulticlassAccuracy: 0.2359583377838135 | Loss: 2.2018291200002036 | Acc: 0.2359583333333333.
Epoch: 4 |

MulticlassAccuracy: 0.2772083282470703 | Loss: 2.1447745552460353 | Acc: 0.2772083333333333.
Epoch: 5 |

MulticlassAccuracy: 0.3231666684150696 | Loss: 2.0870940800507864 | Acc: 0.3231666666666667.
Epoch: 6 |

MulticlassAccuracy: 0.3888333439826965 | Loss: 2.0275067920287451 | Acc: 0.3888333333333333.
Epoch: 7 |

MulticlassAccuracy: 0.4320833384990692 | Loss: 1.9655491570631662 | Acc: 0.4320833333333333.
Epoch: 8 |

MulticlassAccuracy: 0.4537916779518127 | Loss: 1.9023256477912267 | Acc: 0.4537916666666666.
Returned to Spot: Validation loss: 1.9023256477912267

spotPython tuning: 1.030354088718382 [#####] 100.00% Done...

<spotPython.spot.spot.Spot at 0x18220cc40>

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")
```

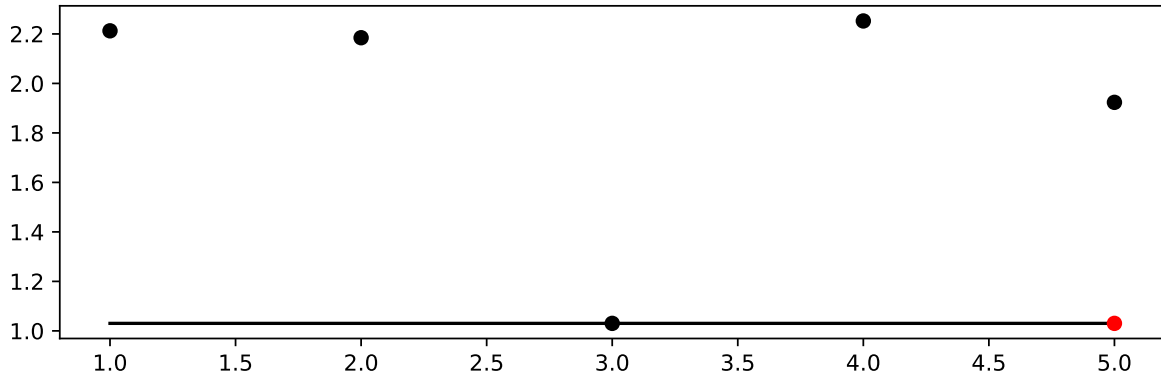


Figure 11.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- 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
```

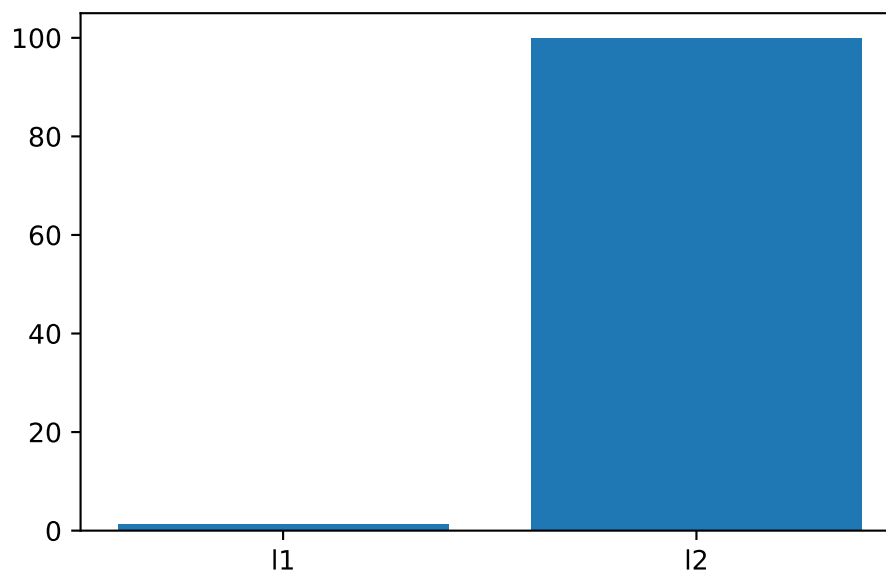


Figure 11.2: Variable importance plot, threshold 0.025.

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: 0.3853749930858612 | Loss: 2.0709688797791799 | Acc: 0.3853750000000000.
Epoch: 2 |

MulticlassAccuracy: 0.5088333487510681 | Loss: 1.6014390341440836 | Acc: 0.5088333333333334.
Epoch: 3 |

MulticlassAccuracy: 0.6063333153724670 | Loss: 1.2798346570730210 | Acc: 0.6063333333333333.
Epoch: 4 |

MulticlassAccuracy: 0.6297083497047424 | Loss: 1.1057451834281287 | Acc: 0.6297083333333333.
Epoch: 5 |

MulticlassAccuracy: 0.6533750295639038 | Loss: 1.0024410256544749 | Acc: 0.6533750000000000.
Epoch: 6 |

MulticlassAccuracy: 0.6658333539962769 | Loss: 0.9345735101103783 | Acc: 0.6658333333333334.
Epoch: 7 |

MulticlassAccuracy: 0.6772083044052124 | Loss: 0.8860317281683286 | Acc: 0.6772083333333333.
Epoch: 8 |

MulticlassAccuracy: 0.6884583234786987 | Loss: 0.8494114318688710 | Acc: 0.6884583333333333.
Returned to Spot: Validation loss: 0.849411431868871

```
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: 0.6773999929428101 | Loss: 0.8630474416732788 | Acc: 0.6774000000000000.
Final evaluation: Validation loss: 0.8630474416732788
Final evaluation: Validation metric: 0.6773999929428101

(0.8630474416732788, nan, tensor(0.6774))

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): 2.239

MulticlassAccuracy: 0.2829999923706055 | Loss: 2.0542246769766015 | Acc: 0.2830000000000000.
Epoch: 2 |

Batch: 10000. Batch Size: 2. Training Loss (running): 1.983

MulticlassAccuracy: 0.4895833432674408 | Loss: 1.7964541525840760 | Acc: 0.4895833333333333.
Epoch: 3 |

Batch: 10000. Batch Size: 2. Training Loss (running): 1.737

MulticlassAccuracy: 0.5982916951179504 | Loss: 1.5727640524258215 | Acc: 0.5982916666666667.
Epoch: 4 |

Batch: 10000. Batch Size: 2. Training Loss (running): 1.521

MulticlassAccuracy: 0.6353750228881836 | Loss: 1.3848808000683785 | Acc: 0.6353750000000000.
Epoch: 5 |

Batch: 10000. Batch Size: 2. Training Loss (running): 1.346

MulticlassAccuracy: 0.6501250267028809 | Loss: 1.2318367388968667 | Acc: 0.6501250000000000.
Epoch: 6 |

Batch: 10000. Batch Size: 2. Training Loss (running): 1.206

MulticlassAccuracy: 0.6654583215713501 | Loss: 1.1092297276332974 | Acc: 0.6654583333333334.
Epoch: 7 |

Batch: 10000. Batch Size: 2. Training Loss (running): 1.096

MulticlassAccuracy: 0.6785416603088379 | Loss: 1.0142268054448069 | Acc: 0.6785416666666667.
Epoch: 8 |

Batch: 10000. Batch Size: 2. Training Loss (running): 0.998

MulticlassAccuracy: 0.6880416870117188 | Loss: 0.9396844933064034 | Acc: 0.6880416666666667.
Returned to Spot: Validation loss: 0.9396844933064034

```
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: 0.6766999959945679 | Loss: 0.9604180929064751 | Acc: 0.6767000000000000.
Final evaluation: Validation loss: 0.9604180929064751
Final evaluation: Validation metric: 0.6766999959945679

(0.9604180929064751, nan, tensor(0.6767))

11.10.5 Detailed Hyperparameter Plots

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

l1: 1.3180004239917138
l2: 100.0

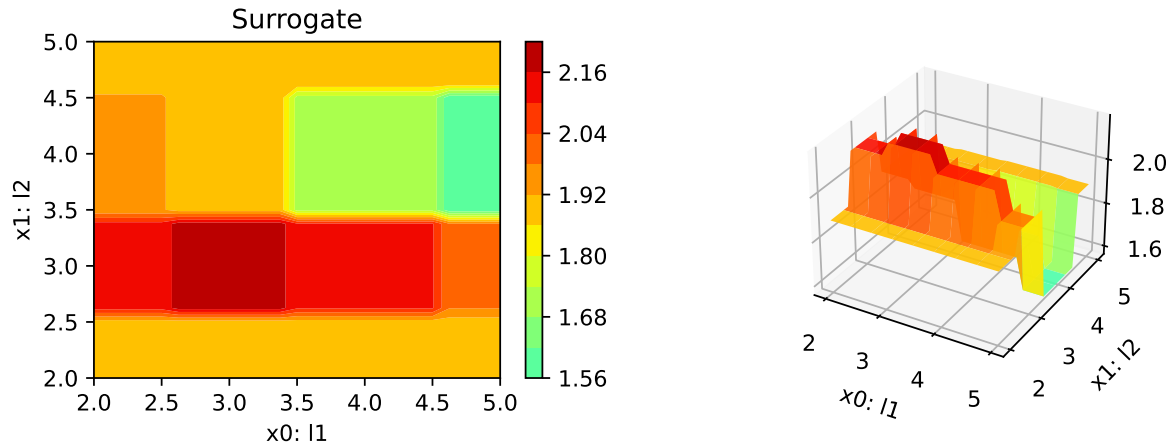


Figure 11.3: Contour plots.

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]"
```

spotPython	0.2.50
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
```

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_2023-06-28_02-10-39

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
```

12.8.3 Starting the Hyperparameter Tuning

[illegible]

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

MulticlassAccuracy: 0.0965000018477440 | Loss: 2.3229436265945433 | Acc: 0.0965000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.0965000018477440 | Loss: 2.3222723131179808 | Acc: 0.0965000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.0965000018477440 | Loss: 2.3215842788696288 | Acc: 0.0965000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.0965000018477440 | Loss: 2.3210316289901733 | Acc: 0.0965000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.0965000018477440 | Loss: 2.3206187932968141 | Acc: 0.0965000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.0965000018477440 | Loss: 2.3202354761123658 | Acc: 0.0965000000000000.
Epoch: 7 |

MulticlassAccuracy: 0.0965000018477440 | Loss: 2.3198463369369509 | Acc: 0.0965000000000000.
Epoch: 8 |

MulticlassAccuracy: 0.0965000018477440 | Loss: 2.3194390724182128 | Acc: 0.0965000000000000.
Returned to Spot: Validation loss: 2.319439072418213

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

MulticlassAccuracy: 0.0947500020265579 | Loss: 2.3300891839027407 | Acc: 0.0947500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.0952999964356422 | Loss: 2.3289024727821350 | Acc: 0.0953000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.0951000005006790 | Loss: 2.3276406299591064 | Acc: 0.0951000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.0944499969482422 | Loss: 2.3264384832382201 | Acc: 0.0944500000000000.
Returned to Spot: Validation loss: 2.32643848323822

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

MulticlassAccuracy: 0.1002999991178513 | Loss: 2.3063549036979674 | Acc: 0.1003000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.1163000017404556 | Loss: 2.2912145838022231 | Acc: 0.1163000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.1983499974012375 | Loss: 2.2703530179500580 | Acc: 0.1983500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.2122000008821487 | Loss: 2.2382009414076807 | Acc: 0.2122000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.2088000029325485 | Loss: 2.2031976449012758 | Acc: 0.2088000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.2117500007152557 | Loss: 2.1712784125328062 | Acc: 0.2117500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.2154500037431717 | Loss: 2.1414270103096964 | Acc: 0.2154500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.2223500013351440 | Loss: 2.1134587463200094 | Acc: 0.2223500000000000.
Returned to Spot: Validation loss: 2.1134587463200094

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

MulticlassAccuracy: 0.1023000031709671 | Loss: 2.3195468776702879 | Acc: 0.1023000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.1017000004649162 | Loss: 2.3185984408378602 | Acc: 0.1017000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.1010499969124794 | Loss: 2.3175690394401549 | Acc: 0.1010500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.1009000018239021 | Loss: 2.3162579026222230 | Acc: 0.1009000000000000.
Returned to Spot: Validation loss: 2.316257902622223

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

MulticlassAccuracy: 0.1002999991178513 | Loss: 2.3069613018035890 | Acc: 0.1003000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.1003499999642372 | Loss: 2.3056407068252565 | Acc: 0.1003500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.1003499999642372 | Loss: 2.3042519001960753 | Acc: 0.1003500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.1003499999642372 | Loss: 2.3026865015983580 | Acc: 0.1003500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.1004000008106232 | Loss: 2.3007477265357972 | Acc: 0.1004000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.1005000025033951 | Loss: 2.2983277581214905 | Acc: 0.1005000000000000.
Epoch: 7 |

MulticlassAccuracy: 0.1062000021338463 | Loss: 2.2953551019668579 | Acc: 0.1062000000000000.
Epoch: 8 |

MulticlassAccuracy: 0.1221999973058701 | Loss: 2.2918088401794434 | Acc: 0.1222000000000000.
Returned to Spot: Validation loss: 2.2918088401794434

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

MulticlassAccuracy: 0.0996000021696091 | Loss: 2.3092764670372010 | Acc: 0.0996000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.0996000021696091 | Loss: 2.3077291632652281 | Acc: 0.0996000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.0996000021696091 | Loss: 2.3055766059875489 | Acc: 0.0996000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.0996000021696091 | Loss: 2.3028528669357300 | Acc: 0.0996000000000000.
Epoch: 5 |

```
MulticlassAccuracy: 0.0996000021696091 | Loss: 2.2998790930747988 | Acc: 0.0996000000000000.  
Epoch: 6 |
```

```
MulticlassAccuracy: 0.0996000021696091 | Loss: 2.2967916171073912 | Acc: 0.0996000000000000.  
Epoch: 7 |
```

```
MulticlassAccuracy: 0.0996500030159950 | Loss: 2.2935000058174135 | Acc: 0.0996500000000000.  
Epoch: 8 |
```

```
MulticlassAccuracy: 0.1006499975919724 | Loss: 2.2900170220375062 | Acc: 0.1006500000000000.  
Returned to Spot: Validation loss: 2.2900170220375062
```

```
spotPython tuning: 2.1134587463200094 [#####] 100.00% Done...
```

```
<spotPython.spot.spot.Spot at 0x17ace3b20>
```

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 `spot_tuner.plot_progress`.

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

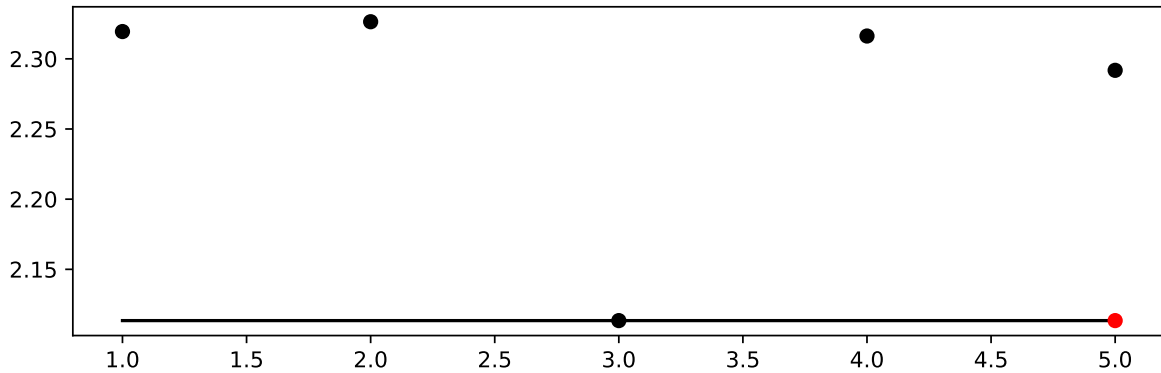


Figure 12.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- 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
```

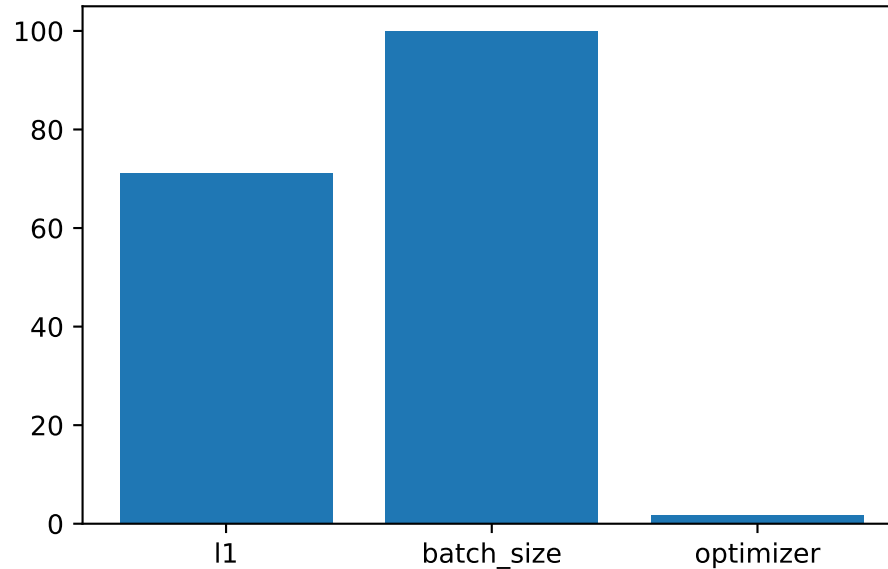


Figure 12.2: Variable importance plot, threshold 0.025.

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): 2.307

MulticlassAccuracy: 0.1030000001192093 | Loss: 2.3013703580141067 | Acc: 0.1030000000000000.
Epoch: 2 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.298

MulticlassAccuracy: 0.1556999981403351 | Loss: 2.2858938544273375 | Acc: 0.1557000000000000.
Epoch: 3 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.278

MulticlassAccuracy: 0.1763000041246414 | Loss: 2.2559600357770919 | Acc: 0.1763000000000000.
Epoch: 4 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.245

MulticlassAccuracy: 0.1843499988317490 | Loss: 2.2157928521037102 | Acc: 0.1843500000000000.
Epoch: 5 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.205

MulticlassAccuracy: 0.1916999965906143 | Loss: 2.1784109441995620 | Acc: 0.1917000000000000.
Epoch: 6 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.174

MulticlassAccuracy: 0.1997999995946884 | Loss: 2.1483312077045440 | Acc: 0.1998000000000000.
Epoch: 7 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.144

MulticlassAccuracy: 0.2089499980211258 | Loss: 2.1242540306627751 | Acc: 0.2089500000000000.
Epoch: 8 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.120

MulticlassAccuracy: 0.2283000051975250 | Loss: 2.1027217387080190 | Acc: 0.2283000000000000.
Returned to Spot: Validation loss: 2.102721738708019

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: 0.2237000018358231 | Loss: 2.0999580399632456 | Acc: 0.2237000000000000.
Final evaluation: Validation loss: 2.0999580399632456
Final evaluation: Validation metric: 0.22370000183582306

(2.0999580399632456, nan, tensor(0.2237))

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-kolds:
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)
```

```
l1: 71.10138855115844
batch_size: 100.0
optimizer: 1.6359974012591116
```

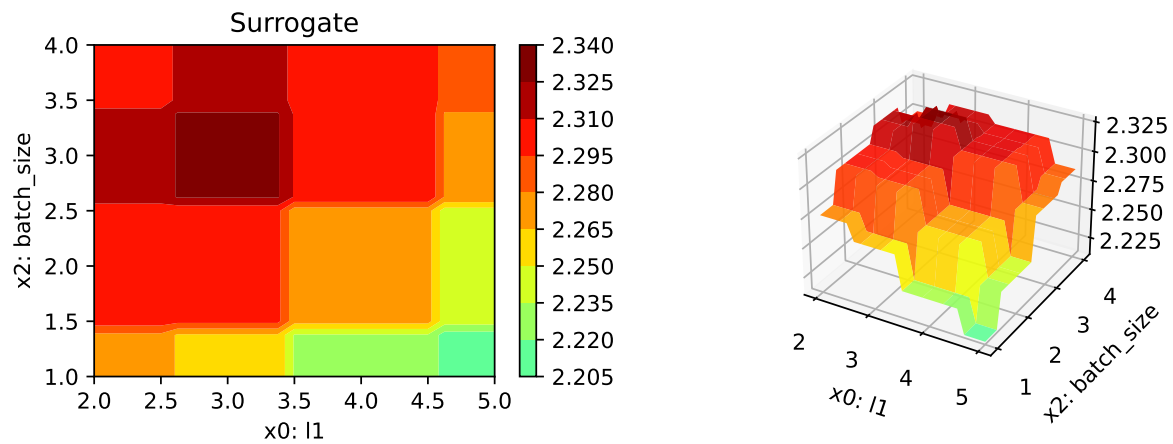
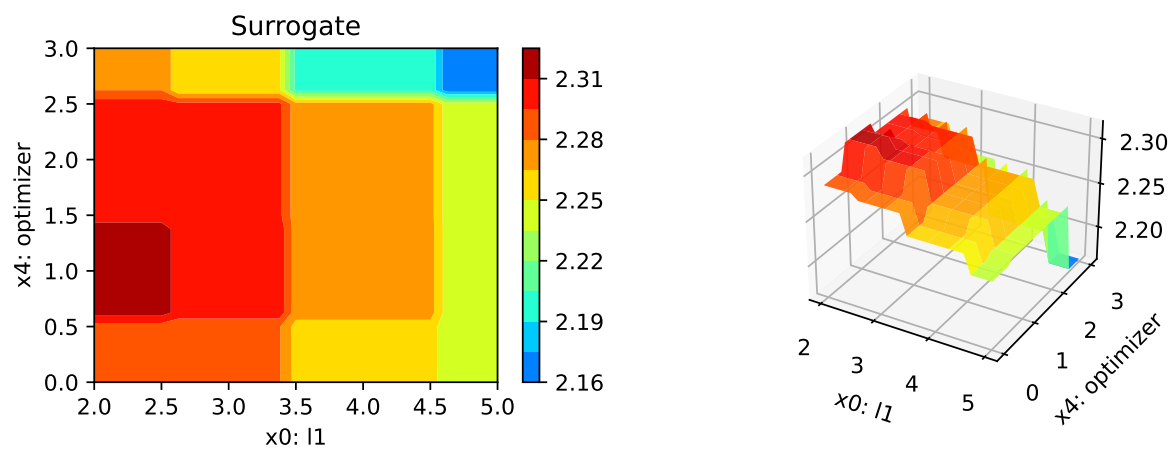
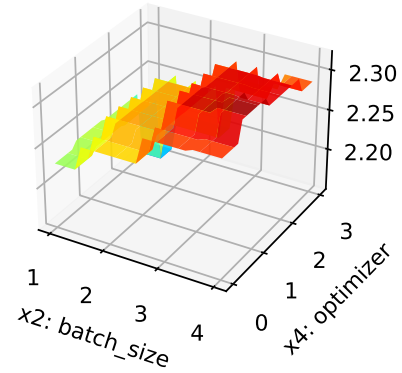
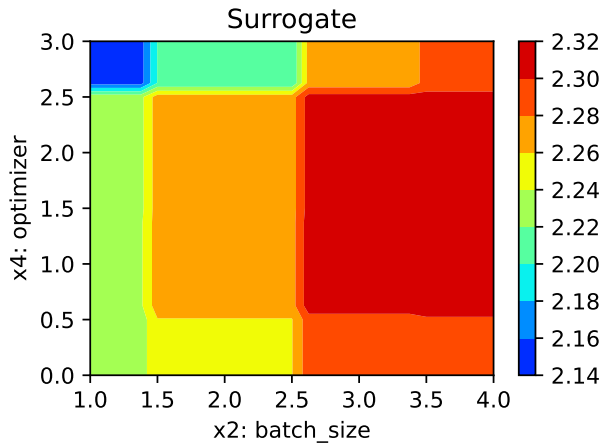


Figure 12.3: Contour plots.





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_2023-06-28_02-39-08

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          0.2.50
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: 2.1863477084825855 [#####----] 57.74%

spotPython tuning: 2.1863477084825855 [#####] 100.00% Done...

<spotPython.spot.spot.Spot at 0x17f51b2b0>

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)

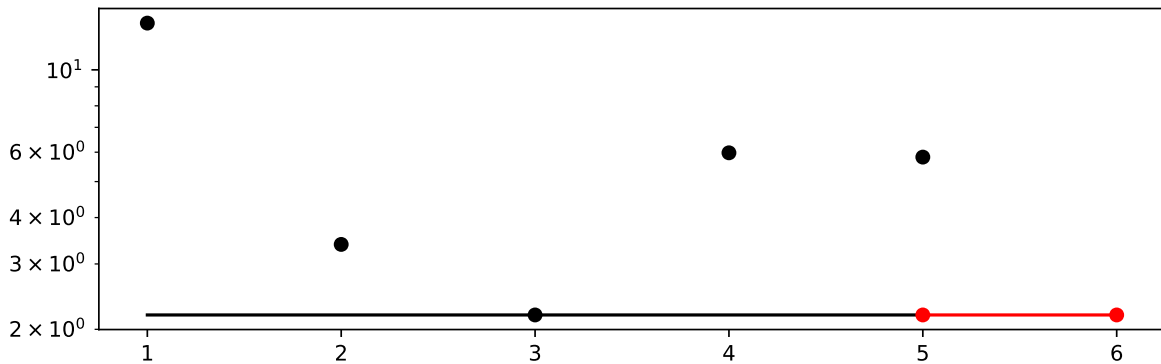
if LOAD:
    result_file_name = "res_ch10-friedman-hpt-0_maans03_60min_20init_1K_2023-04-14_10-11-1

```

```
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

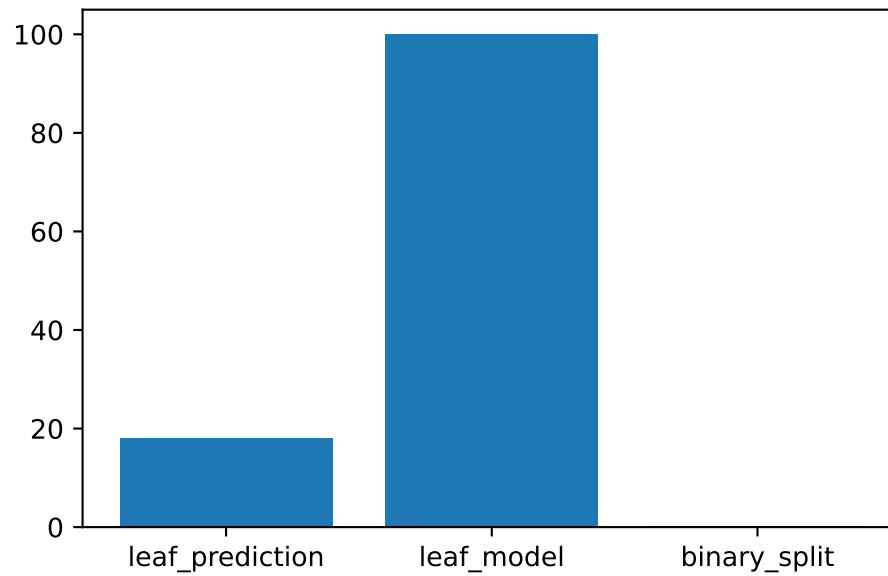
```
print(gen_design_table(fun_control=fun_control, spot=spot_tuner))
```

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

```

```

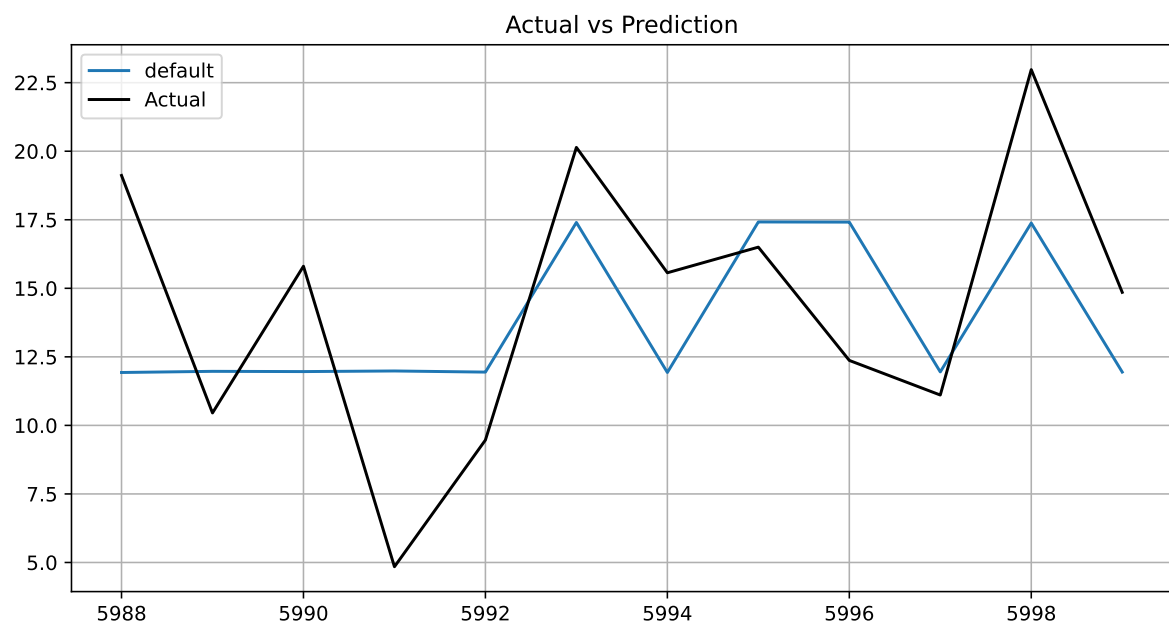
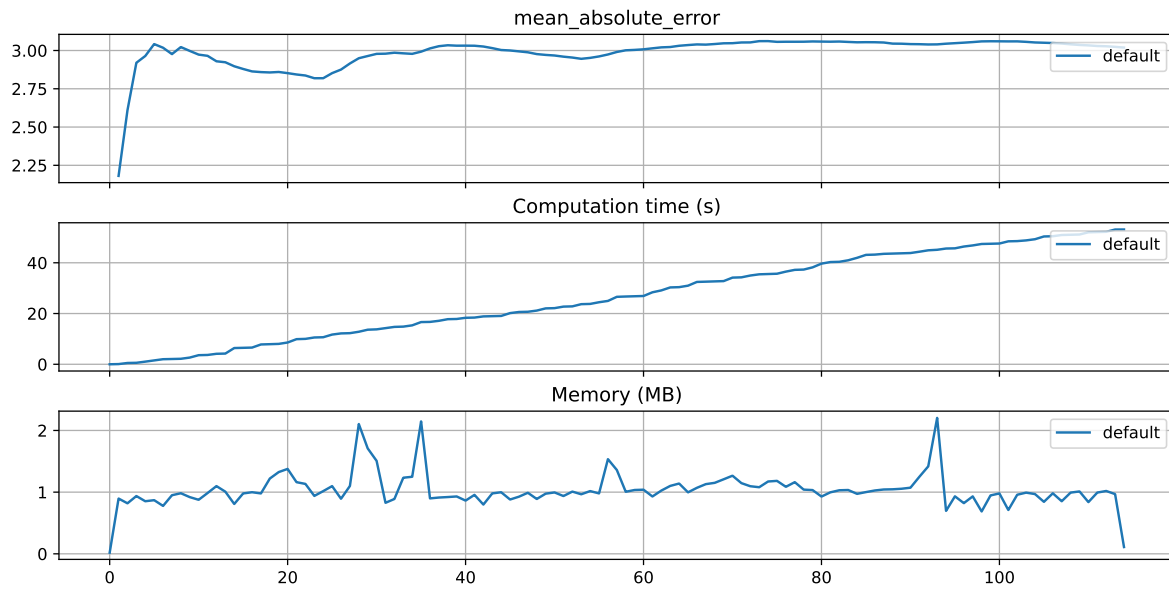
df_eval_default, df_true_default = eval_oml_horizon(
    model=model_default,
    train=fun_control["train"],
    test=fun_control["test"],
    target_column=fun_control["target_column"],
    horizon=fun_control["horizon"],
    oml_grace_period=fun_control["oml_grace_period"],
    metric=fun_control["metric_sklearn"],
)

```

```

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
)

```

```

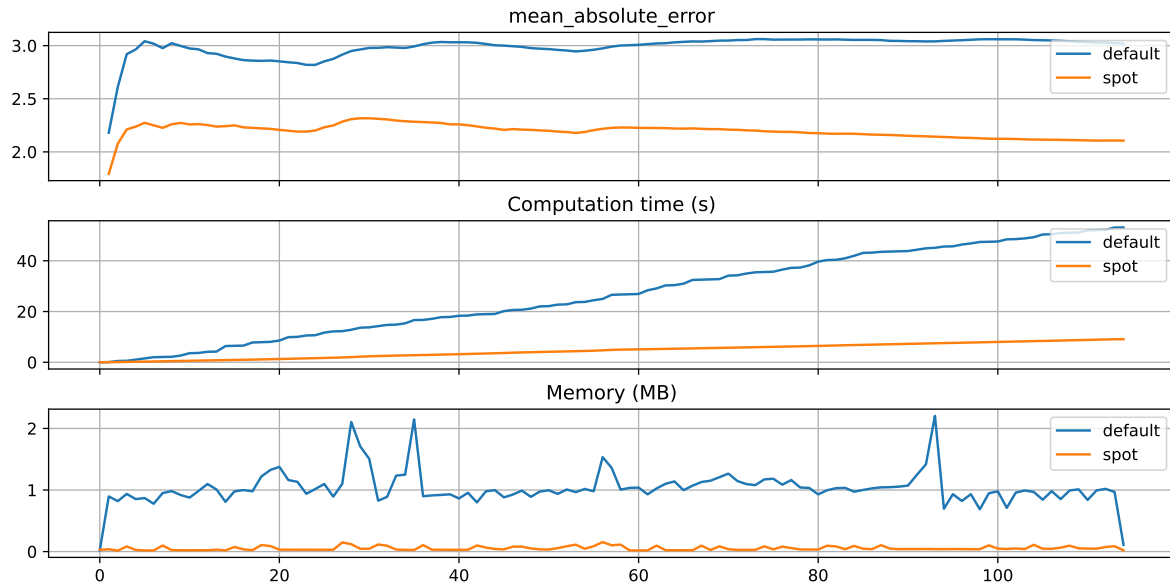
df_eval_spot, df_true_spot = eval_oml_horizon(
    model=model_spot,
    train=fun_control["train"],
    test=fun_control["test"],
    target_column=fun_control["target_column"],
    horizon=fun_control["horizon"],
    oml_grace_period=fun_control["oml_grace_period"],
    metric=fun_control["metric_sklearn"],
)

```

```

df_labels=["default", "spot"]
plot_bml_oml_horizon_metrics(df_eval = [df_eval_default, df_eval_spot], log_y=False, df_la

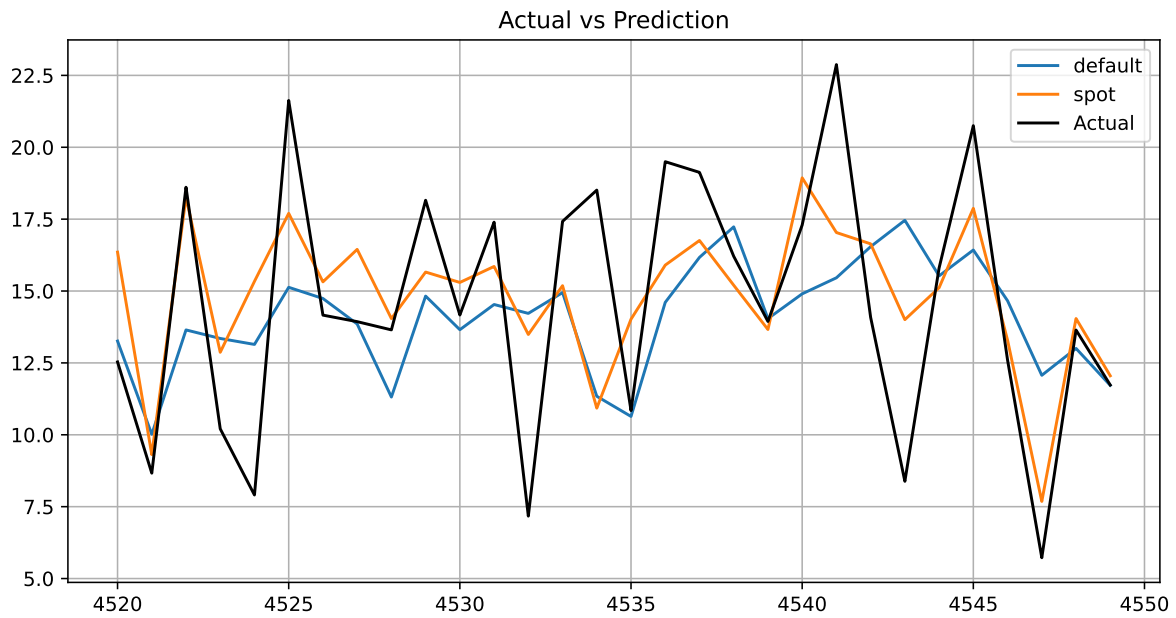
```



```

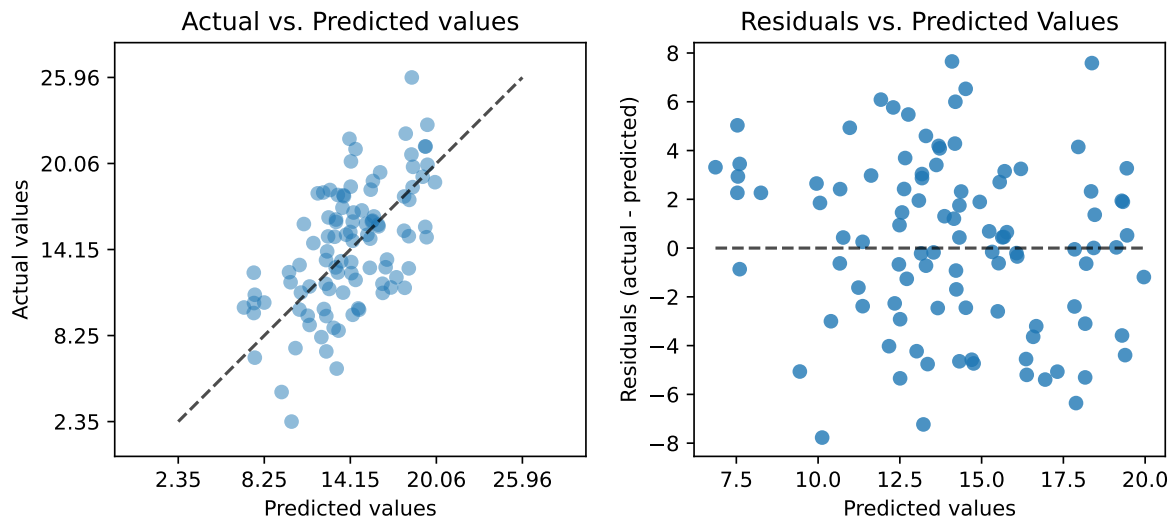
a = int(m/2)+20
b = int(m/2)+50
plot_bml_oml_horizon_predictions(df_true = [df_true_default[a:b], df_true_spot[a:b]], targ

```

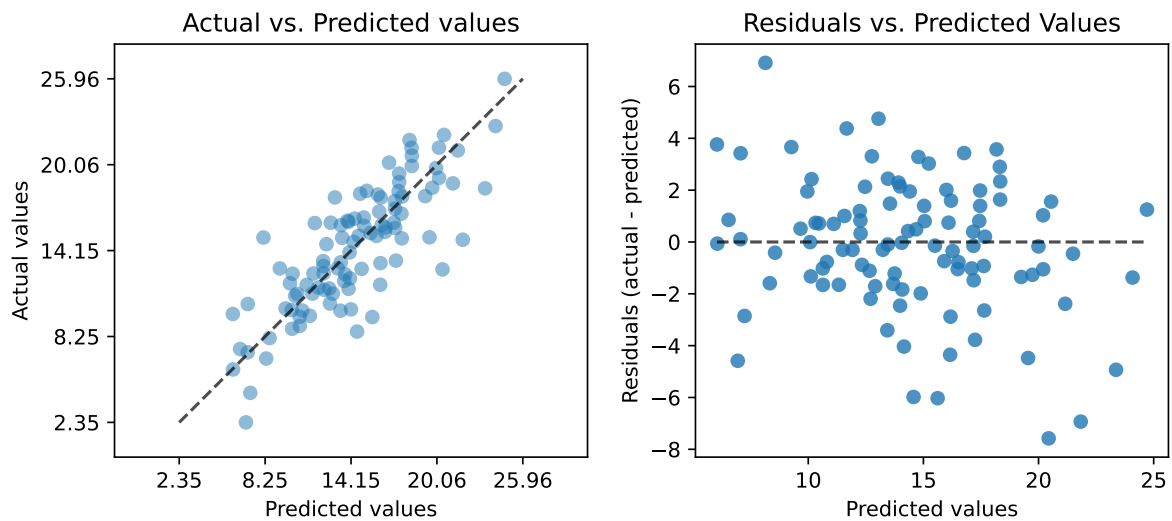


```
from spotPython.plot.validation import plot_actual_vs_predicted
plot_actual_vs_predicted(y_test=df_true_default["y"], y_pred=df_true_default["Prediction"])
plot_actual_vs_predicted(y_test=df_true_spot["y"], y_pred=df_true_spot["Prediction"], titl
```

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': 23,  
 'n_branches': 11,  
 'n_leaves': 12,  
 'n_active_leaves': 30,  
 'n_inactive_leaves': 0,  
 'height': 6,  
 'total_observed_weight': 39002.0,  
 'n_alternate_trees': 27,  
 'n_pruned_alternate_trees': 18,  
 '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	23
n_branches	17	11
n_leaves	18	12
n_active_leaves	96	30
n_inactive_leaves	0	0
height	6	6
total_observed_weight	39002	39002
n_alternate_trees	21	27
n_pruned_alternate_trees	6	18
n_switch_alternate_trees	2	2

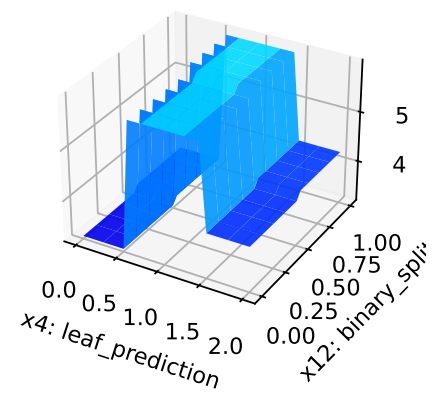
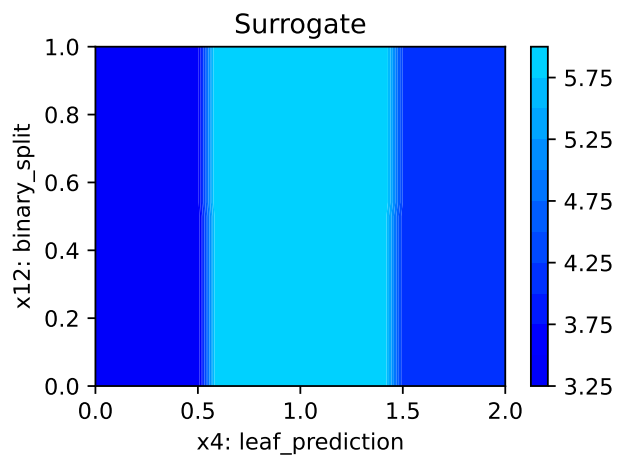
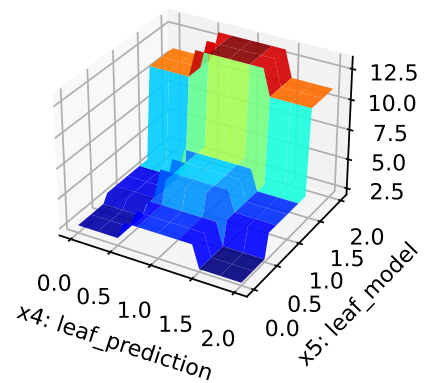
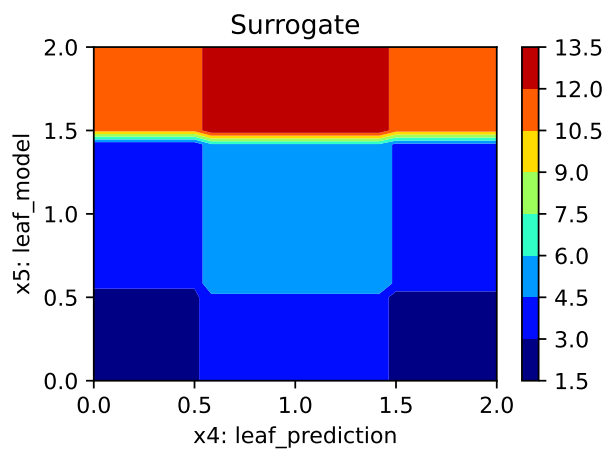
```
min(spot_tuner.y), max(spot_tuner.y)
```

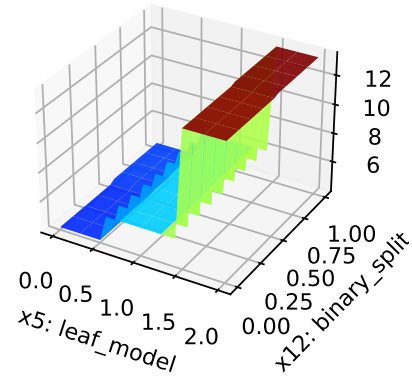
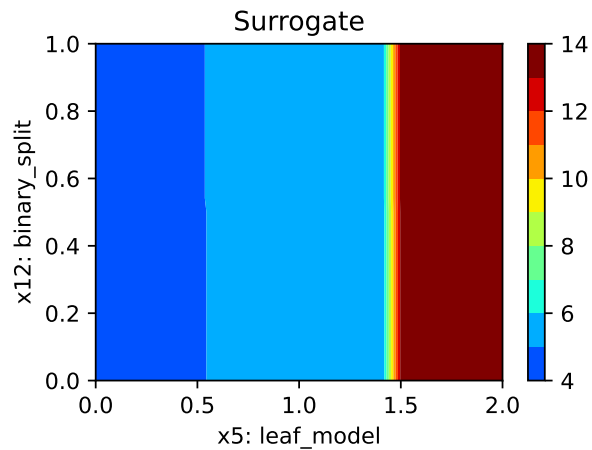
```
(2.1863477084825855, 13.36277370730995)
```

13.9.8 Detailed Hyperparameter Plots

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

```
leaf_prediction: 18.01546615262282  
leaf_model: 100.0  
binary_split: 0.07496781087550164
```





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>	0.2.50
<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`.


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

¹Alternatively, the source code can be downloaded from `gitHub`: <https://github.com/sequential-parameter-optimization/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

```
import os
import copy
import socket
```

²We were not able to install Ray Tune on our system. Therefore, we used the results from the PyTorch tutorial.

```

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_2023-06-28_03-24-10

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, 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
```

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" lambd, "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.	-	-	-	-	-	a	-	-	-	-	-
AdamW	1e-3	-	1e-2	-	-	-	-	-	a	-	-	-	-	-
SparseAdam	1e-3	-	-	-	-	-	-	-	a	-	-	-	-	-
Adamax	2e-3	-	0.	-	-	-	-	-	a	-	-	-	-	-

Optimizer	lr	m	w_d	d	n	r	l_s	l_d	b	l	a	m_d	e	s_s
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.

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.

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 22.14 (in the Appendix).

Table 14.2: Overview of the evaluation settings.

eval	train	test	function	comment
"train_hold_out" ✓			train_one_epoch(), validate_one_epoch() for early stopping	splits the train data set internally
"test_hold_out" ✓	✓	✓	train_one_epoch(), validate_one_epoch() for early stopping	use the test data set for validate_one_epoch()
"train_cv" ✓	✓		evaluate_cv(net, train)	CV using the train data set
"test_cv"		✓	evaluate_cv(net, test)	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 [22.14.1.4](#) 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. `gen_design_table` 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.

```
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)
```

```

config: {'l1': 128, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 32, 'epochs': 16, 'k_folds': 0, 'j
Epoch: 1 |

```

```

MulticlassAccuracy: 0.4000999927520752 | Loss: 1.6513567115783692 | Acc: 0.4001000000000000.
Epoch: 2 |

```

```

MulticlassAccuracy: 0.4535999894142151 | Loss: 1.5048331209182739 | Acc: 0.4536000000000000.
Epoch: 3 |

```

```

MulticlassAccuracy: 0.4765500128269196 | Loss: 1.4416672352790834 | Acc: 0.4765500000000000.
Epoch: 4 |

```

MulticlassAccuracy: 0.5089499950408936 | Loss: 1.3743622310638428 | Acc: 0.5089500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5306000113487244 | Loss: 1.3234944149017334 | Acc: 0.5306000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5401999950408936 | Loss: 1.3090167645454407 | Acc: 0.5402000000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5479500293731689 | Loss: 1.2833861721038817 | Acc: 0.5479500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5641000270843506 | Loss: 1.2293452502250672 | Acc: 0.5641000000000000.
Epoch: 9 |

MulticlassAccuracy: 0.5632500052452087 | Loss: 1.2525316659450532 | Acc: 0.5632500000000000.
Epoch: 10 |

MulticlassAccuracy: 0.5764999985694885 | Loss: 1.2125522743225097 | Acc: 0.5765000000000000.
Epoch: 11 |

MulticlassAccuracy: 0.5727499723434448 | Loss: 1.2419614564895629 | Acc: 0.5727500000000000.
Epoch: 12 |

MulticlassAccuracy: 0.5679000020027161 | Loss: 1.2483667596817016 | Acc: 0.5679000000000000.
Epoch: 13 |

MulticlassAccuracy: 0.5752999782562256 | Loss: 1.2778865601539611 | Acc: 0.5753000000000000.
Early stopping at epoch 12
Returned to Spot: Validation loss: 1.277886560153961

config: {'l1': 16, 'l2': 16, 'lr_mult': 1.0, 'batch_size': 8, 'epochs': 8, 'k_folds': 0, 'pa
Epoch: 1 |

MulticlassAccuracy: 0.4483500123023987 | Loss: 1.5142917092323303 | Acc: 0.4483500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.5042999982833862 | Loss: 1.3780224430441856 | Acc: 0.5043000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.5218499898910522 | Loss: 1.3419508804559708 | Acc: 0.5218500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5383999943733215 | Loss: 1.3139836833834648 | Acc: 0.5384000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5379499793052673 | Loss: 1.2997473927140235 | Acc: 0.5379500000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5322999954223633 | Loss: 1.3622919375538827 | Acc: 0.5323000000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5560500025749207 | Loss: 1.2707115814507008 | Acc: 0.5560500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5682500004768372 | Loss: 1.2285627397835255 | Acc: 0.5682500000000000.
Returned to Spot: Validation loss: 1.2285627397835255

config: {'l1': 256, 'l2': 128, 'lr_mult': 1.0, 'batch_size': 2, 'epochs': 16, 'k_folds': 0,
Epoch: 1 |

MulticlassAccuracy: 0.0970999971032143 | Loss: 2.3100663211107255 | Acc: 0.0971000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.1015999987721443 | Loss: 2.3057759776830675 | Acc: 0.1016000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.0970999971032143 | Loss: 2.3097578743696214 | Acc: 0.0971000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.1010999977588654 | Loss: 2.3064621152639391 | Acc: 0.1011000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.1010999977588654 | Loss: 2.3057633457660676 | Acc: 0.1011000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.0970999971032143 | Loss: 2.3123365935325624 | Acc: 0.0971000000000000.
Epoch: 7 |

MulticlassAccuracy: 0.0999500006437302 | Loss: 2.3104944870710371 | Acc: 0.0999500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.0986000001430511 | Loss: 2.3044364910840986 | Acc: 0.0986000000000000.
Epoch: 9 |

MulticlassAccuracy: 0.0987000018358231 | Loss: 2.3061569568634033 | Acc: 0.0987000000000000.
Epoch: 10 |

MulticlassAccuracy: 0.0998999997973442 | Loss: 2.3067967734813690 | Acc: 0.0999000000000000.
Epoch: 11 |

MulticlassAccuracy: 0.0970999971032143 | Loss: 2.3060884963035582 | Acc: 0.0971000000000000.
Early stopping at epoch 10
Returned to Spot: Validation loss: 2.306088496303558

config: {'l1': 8, 'l2': 32, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pat.
Epoch: 1 |

MulticlassAccuracy: 0.3573000133037567 | Loss: 1.7124652969121934 | Acc: 0.3573000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.4309999942779541 | Loss: 1.5244223443627358 | Acc: 0.4310000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.4622499942779541 | Loss: 1.4652285068541766 | Acc: 0.4622500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.4779500067234039 | Loss: 1.4291896461457014 | Acc: 0.4779500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5113000273704529 | Loss: 1.3385992901623249 | Acc: 0.5113000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5214499831199646 | Loss: 1.3133301689285786 | Acc: 0.5214500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5127000212669373 | Loss: 1.3372075770795346 | Acc: 0.5127000000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5162500143051147 | Loss: 1.3666886946611105 | Acc: 0.5162500000000000.
Returned to Spot: Validation loss: 1.3666886946611105

config: {'l1': 64, 'l2': 512, 'lr_mult': 1.0, 'batch_size': 16, 'epochs': 16, 'k_folds': 0,
Epoch: 1 |

MulticlassAccuracy: 0.4580000042915344 | Loss: 1.4705735107898712 | Acc: 0.4580000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.4979499876499176 | Loss: 1.3791324110984802 | Acc: 0.4979500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.5218499898910522 | Loss: 1.3298448552131652 | Acc: 0.5218500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5341500043869019 | Loss: 1.2997835334777832 | Acc: 0.5341500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5461500287055969 | Loss: 1.2713028556346893 | Acc: 0.5461500000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5521000027656555 | Loss: 1.2497051509380341 | Acc: 0.5521000000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5598499774932861 | Loss: 1.2376025533199311 | Acc: 0.5598500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5630499720573425 | Loss: 1.2263639355659486 | Acc: 0.5630500000000001.
Epoch: 9 |

MulticlassAccuracy: 0.5677000284194946 | Loss: 1.2163668330669404 | Acc: 0.5677000000000000.
Epoch: 10 |

MulticlassAccuracy: 0.5692999958992004 | Loss: 1.2036730331420900 | Acc: 0.5693000000000000.
Epoch: 11 |

MulticlassAccuracy: 0.5774000287055969 | Loss: 1.1997340130329133 | Acc: 0.5774000000000000.
Epoch: 12 |

MulticlassAccuracy: 0.5770000219345093 | Loss: 1.1952938462018967 | Acc: 0.5770000000000000.
Epoch: 13 |

MulticlassAccuracy: 0.5813999772071838 | Loss: 1.1880820136785508 | Acc: 0.5814000000000000.
Epoch: 14 |

MulticlassAccuracy: 0.5856500267982483 | Loss: 1.1798746233463286 | Acc: 0.5856500000000000.
Epoch: 15 |

MulticlassAccuracy: 0.5871000289916992 | Loss: 1.1753327502250672 | Acc: 0.5871000000000000.
Epoch: 16 |

MulticlassAccuracy: 0.5885499715805054 | Loss: 1.1664345373392104 | Acc: 0.5885500000000000.
Returned to Spot: Validation loss: 1.1664345373392104

config: {'l1': 64, 'l2': 256, 'lr_mult': 1.0, 'batch_size': 16, 'epochs': 16, 'k_folds': 0,
Epoch: 1 |

MulticlassAccuracy: 0.4467000067234039 | Loss: 1.5021803729534149 | Acc: 0.4467000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.4828499853610992 | Loss: 1.4189377374172212 | Acc: 0.4828500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.5012999773025513 | Loss: 1.3753459608554841 | Acc: 0.5013000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5175999999046326 | Loss: 1.3321617524623870 | Acc: 0.5175999999999999.
Epoch: 5 |

MulticlassAccuracy: 0.5271000266075134 | Loss: 1.3109904826164245 | Acc: 0.5271000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5387499928474426 | Loss: 1.2865047058820724 | Acc: 0.5387500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5447000265121460 | Loss: 1.2728297123908996 | Acc: 0.5447000000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5519000291824341 | Loss: 1.2613665298938752 | Acc: 0.5518999999999999.
Epoch: 9 |

MulticlassAccuracy: 0.5572999715805054 | Loss: 1.2420495476245881 | Acc: 0.5573000000000000.
Epoch: 10 |

MulticlassAccuracy: 0.5597500205039978 | Loss: 1.2366037570953370 | Acc: 0.5597500000000000.
Epoch: 11 |

MulticlassAccuracy: 0.5660499930381775 | Loss: 1.2192751317501067 | Acc: 0.5660500000000001.
Epoch: 12 |

MulticlassAccuracy: 0.5685499906539917 | Loss: 1.2155537758350372 | Acc: 0.5685500000000000.
Epoch: 13 |

MulticlassAccuracy: 0.5705000162124634 | Loss: 1.2095361758947372 | Acc: 0.5705000000000000.
Epoch: 14 |

MulticlassAccuracy: 0.5733000040054321 | Loss: 1.2013895448684693 | Acc: 0.5733000000000000.
Epoch: 15 |

MulticlassAccuracy: 0.5793499946594238 | Loss: 1.1944043334722518 | Acc: 0.5793500000000000.
Epoch: 16 |

MulticlassAccuracy: 0.5812500119209290 | Loss: 1.1896025751590729 | Acc: 0.5812500000000000.
Returned to Spot: Validation loss: 1.1896025751590729

spotPython tuning: 1.1664345373392104 [####-----] 36.33%

config: {'l1': 64, 'l2': 128, 'lr_mult': 1.0, 'batch_size': 2, 'epochs': 8, 'k_folds': 0, 'p': 0.5}
Epoch: 1 |

MulticlassAccuracy: 0.4776999950408936 | Loss: 1.4998574424954305 | Acc: 0.4777000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.5234500169754028 | Loss: 1.4802662259458605 | Acc: 0.5234500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.5397999882698059 | Loss: 1.4099641055857255 | Acc: 0.5397999999999999.
Epoch: 4 |

MulticlassAccuracy: 0.5354999899864197 | Loss: 1.5541547392516173 | Acc: 0.5355000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5526499748229980 | Loss: 1.5814915931325360 | Acc: 0.5526500000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5562999844551086 | Loss: 1.5380642236348019 | Acc: 0.5563000000000000.
Early stopping at epoch 5
Returned to Spot: Validation loss: 1.5380642236348019

spotPython tuning: 1.1664345373392104 [#####-] 92.96%

config: {'l1': 64, 'l2': 256, 'lr_mult': 1.0, 'batch_size': 16, 'epochs': 16, 'k_folds': 0,

Epoch: 1 |

MulticlassAccuracy: 0.460299985218048 | Loss: 1.4725294163227081 | Acc: 0.4603000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.5149999856948853 | Loss: 1.3508248186826706 | Acc: 0.5150000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.5278999805450439 | Loss: 1.3024248506069183 | Acc: 0.5279000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5498999953269958 | Loss: 1.2531586042165757 | Acc: 0.5499000000000001.
Epoch: 5 |

MulticlassAccuracy: 0.5800999999046326 | Loss: 1.1794013554573060 | Acc: 0.5800999999999999.
Epoch: 6 |

MulticlassAccuracy: 0.5819500088691711 | Loss: 1.1868921639442445 | Acc: 0.5819500000000000.
Epoch: 7 |

```
MulticlassAccuracy: 0.5817999839782715 | Loss: 1.1813738886713983 | Acc: 0.5818000000000000.  
Epoch: 8 |
```

```
MulticlassAccuracy: 0.6023499965667725 | Loss: 1.1577861665248872 | Acc: 0.6023500000000001.  
Epoch: 9 |
```

```
MulticlassAccuracy: 0.5963000059127808 | Loss: 1.1838473379850387 | Acc: 0.5963000000000001.  
Epoch: 10 |
```

```
MulticlassAccuracy: 0.5787000060081482 | Loss: 1.2445974563360214 | Acc: 0.5787000000000000.  
Epoch: 11 |
```

```
MulticlassAccuracy: 0.5993000268936157 | Loss: 1.1878549308300019 | Acc: 0.5993000000000001.  
Early stopping at epoch 10  
Returned to Spot: Validation loss: 1.187854930830002
```

```
spotPython tuning: 1.1664345373392104 [#####] 100.00% Done...
```

```
<spotPython.spot.spot.Spot at 0x1858f1b40>
```

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.

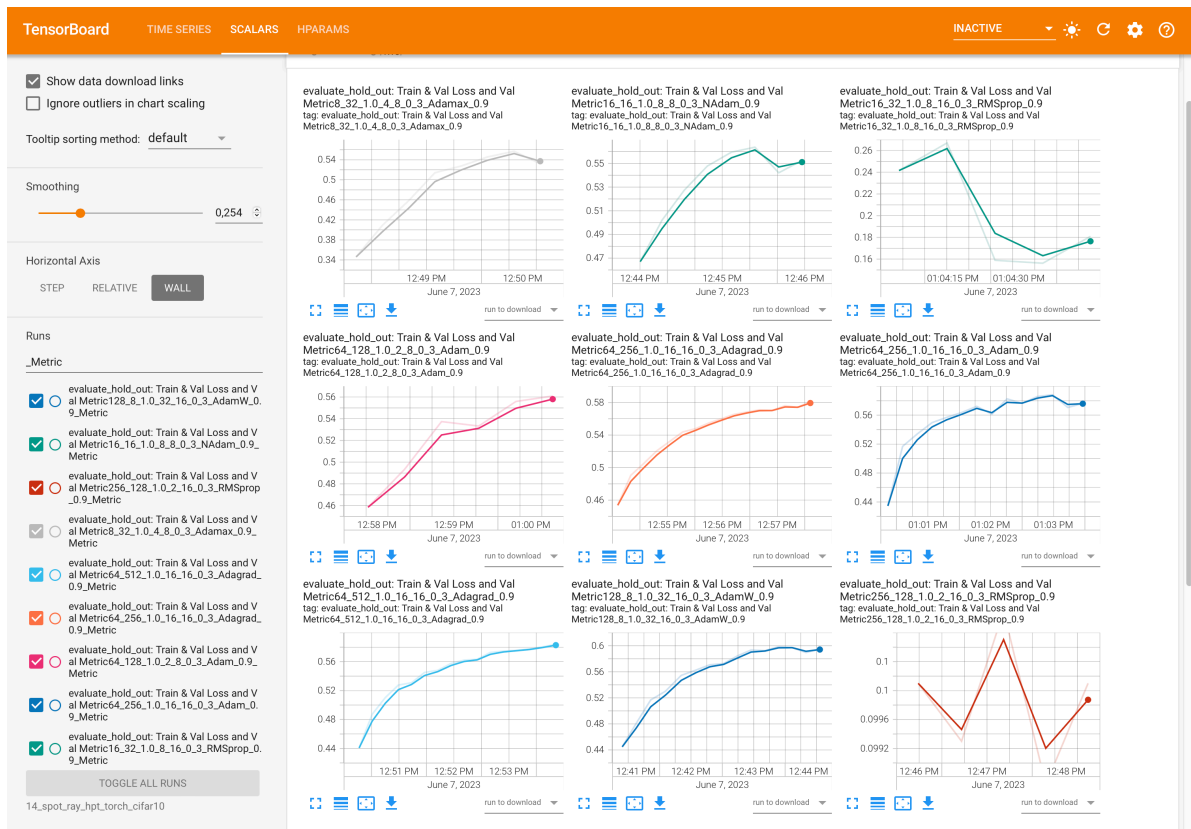


Figure 14.1: Tensorboard

TensorBoard									
INACTIVE									
TABLE VIEW									
Trial ID	Show Metrics	l1	l2	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")
```

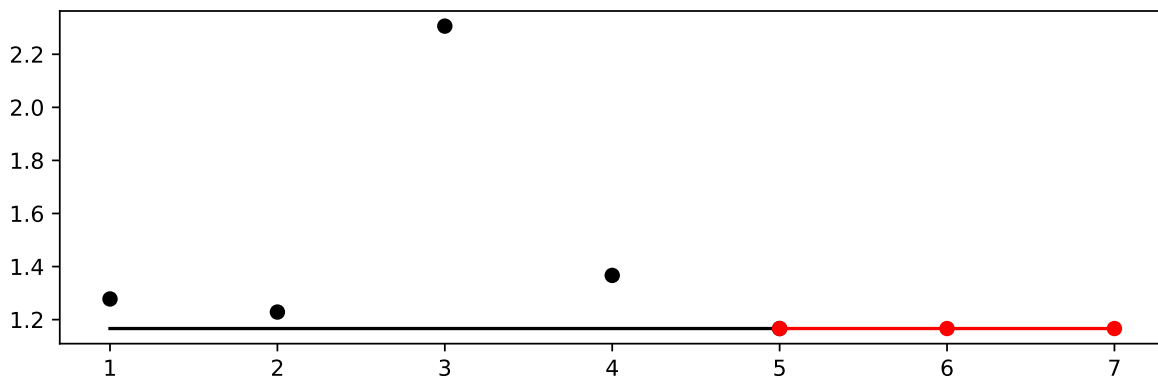


Figure 14.3: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

?@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-optimization with the surrogate 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	6.0	transform_power_2_int
l2	int	5	2.0	9.0	9.0	transform_power_2_int
lr_mult	float	1.0	1.0	1.0	1.0	None
batch_size	int	4	1.0	5.0	4.0	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	1.0	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")
```

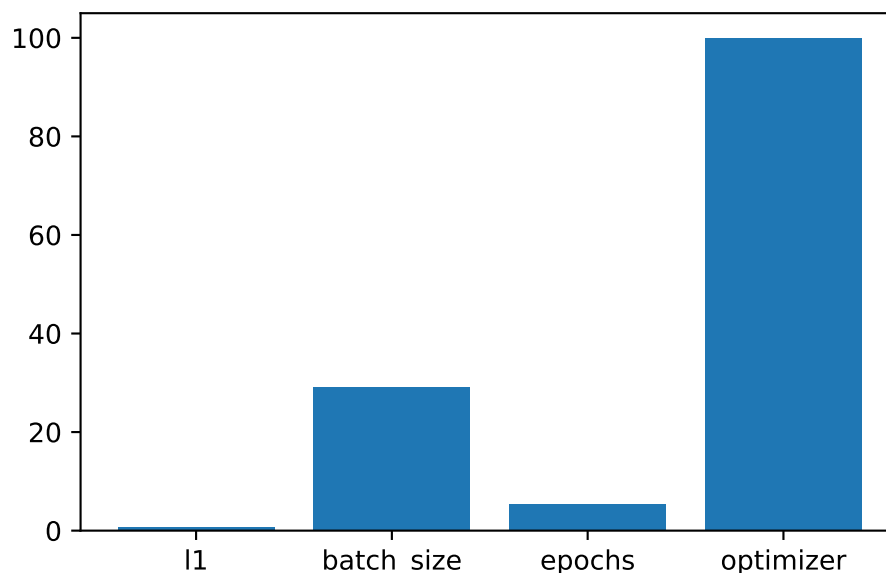



Figure 14.4: Variable importance plot, threshold 0.025.

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=64, bias=True)
  (fc2): Linear(in_features=64, out_features=512, bias=True)
  (fc3): Linear(in_features=512, 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: 0.1014999970793724 | Loss: 2.3019537027359007 | Acc: 0.1015000000000000.

Epoch: 2 |

MulticlassAccuracy: 0.1253499984741211 | Loss: 2.2974282182693480 | Acc: 0.1253500000000000.

Epoch: 3 |

MulticlassAccuracy: 0.1172000020742416 | Loss: 2.2871326692581175 | Acc: 0.1172000000000000.

Epoch: 4 |

MulticlassAccuracy: 0.1555500030517578 | Loss: 2.2574999176025390 | Acc: 0.1555500000000000.

Epoch: 5 |

MulticlassAccuracy: 0.1994500011205673 | Loss: 2.2153904356002809 | Acc: 0.1994500000000000.

Epoch: 6 |

MulticlassAccuracy: 0.2230000048875809 | Loss: 2.1780399012565614 | Acc: 0.2230000000000000.

Epoch: 7 |

MulticlassAccuracy: 0.2379499971866608 | Loss: 2.1401176103591917 | Acc: 0.2379500000000000.

Epoch: 8 |

MulticlassAccuracy: 0.2468499988317490 | Loss: 2.1081568152427672 | Acc: 0.2468500000000000.

Returned to Spot: Validation loss: 2.108156815242767

MulticlassAccuracy: 0.2477999925613403 | Loss: 2.1077489068984985 | Acc: 0.2478000000000000.

Final evaluation: Validation loss: 2.1077489068984985

Final evaluation: Validation metric: 0.24779999256134033

(2.1077489068984985, nan, tensor(0.2478))

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: 0.4679000079631805 | Loss: 1.4511152942657470 | Acc: 0.4679000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.5044999718666077 | Loss: 1.3676464962482453 | Acc: 0.5044999999999999.
Epoch: 3 |

MulticlassAccuracy: 0.5162500143051147 | Loss: 1.3479596774101257 | Acc: 0.5162500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5327000021934509 | Loss: 1.2912749471664429 | Acc: 0.5327000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5465999841690063 | Loss: 1.2642055624008179 | Acc: 0.5466000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5530999898910522 | Loss: 1.2610540261745453 | Acc: 0.5531000000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5655500292778015 | Loss: 1.2186668899059296 | Acc: 0.5655500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5727000236511230 | Loss: 1.1998199648141861 | Acc: 0.5727000000000000.
Epoch: 9 |

MulticlassAccuracy: 0.5763999819755554 | Loss: 1.1890307539939879 | Acc: 0.5764000000000000.
Epoch: 10 |

MulticlassAccuracy: 0.5788000226020813 | Loss: 1.1902390249729156 | Acc: 0.5788000000000000.
Epoch: 11 |

MulticlassAccuracy: 0.5830500125885010 | Loss: 1.1716840088367462 | Acc: 0.5830500000000000.
Epoch: 12 |

MulticlassAccuracy: 0.5874000191688538 | Loss: 1.1752234544515610 | Acc: 0.5874000000000000.
Epoch: 13 |

MulticlassAccuracy: 0.5909000039100647 | Loss: 1.1566826323747634 | Acc: 0.5909000000000000.
Epoch: 14 |

MulticlassAccuracy: 0.5922999978065491 | Loss: 1.1544693437099456 | Acc: 0.5923000000000000.
Epoch: 15 |

MulticlassAccuracy: 0.5950000286102295 | Loss: 1.1510410333156587 | Acc: 0.5950000000000000.
Epoch: 16 |

MulticlassAccuracy: 0.5969499945640564 | Loss: 1.1403570478439331 | Acc: 0.5969500000000000.
Returned to Spot: Validation loss: 1.140357047843933

MulticlassAccuracy: 0.5957999825477600 | Loss: 1.1496567366600037 | Acc: 0.5958000000000000.
Final evaluation: Validation loss: 1.1496567366600037
Final evaluation: Validation metric: 0.59579998254776

(1.1496567366600037, nan, tensor(0.5958))

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: 0.7932637713169925
batch_size: 29.082362374943912
epochs: 5.37233422887797
optimizer: 100.0
```

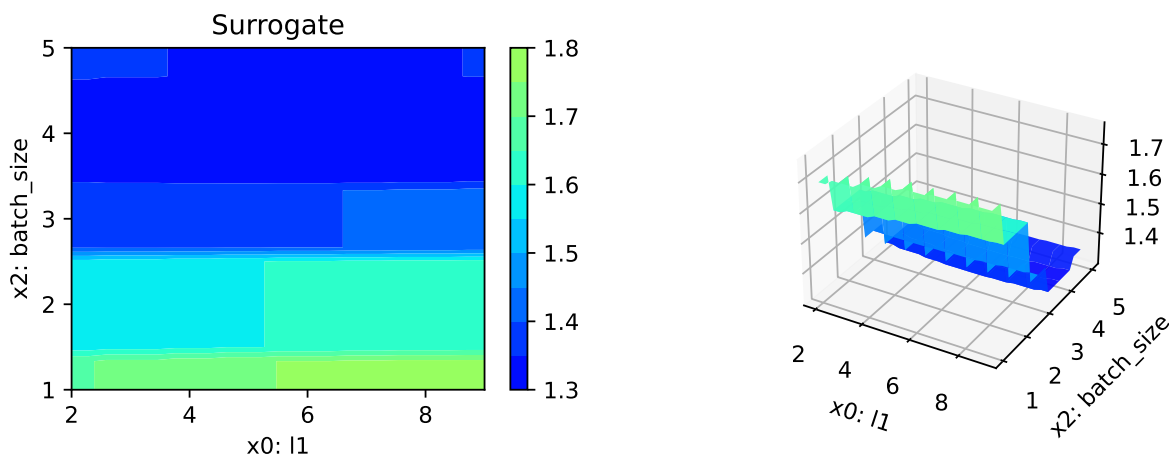
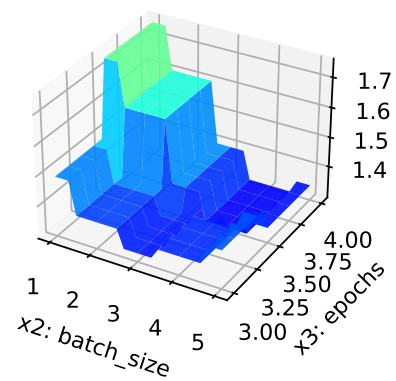
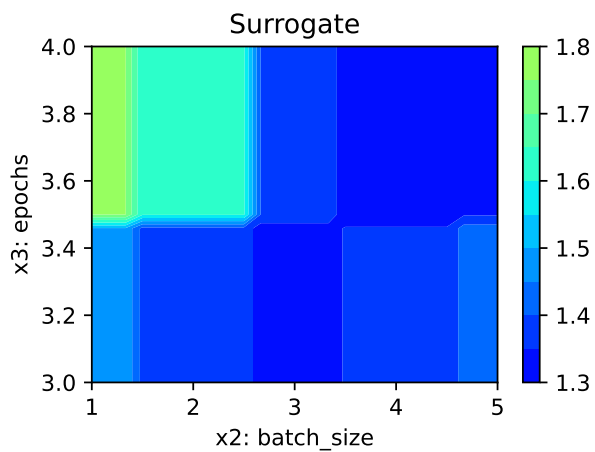
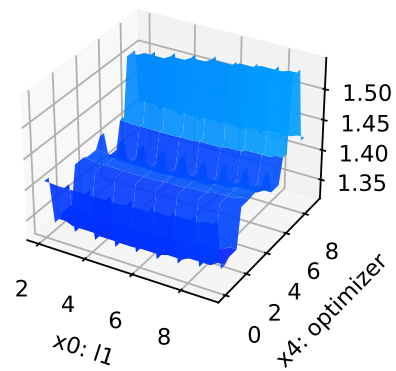
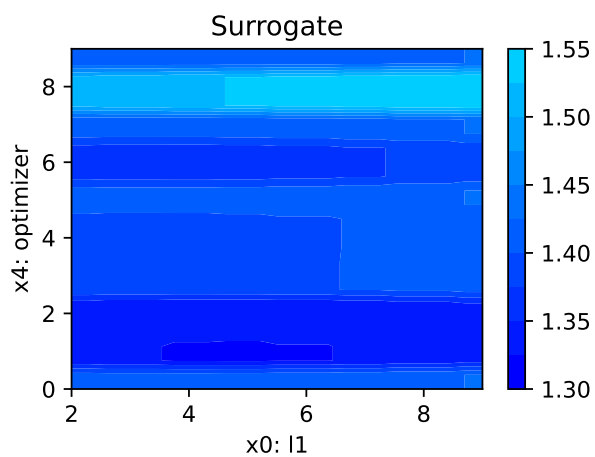
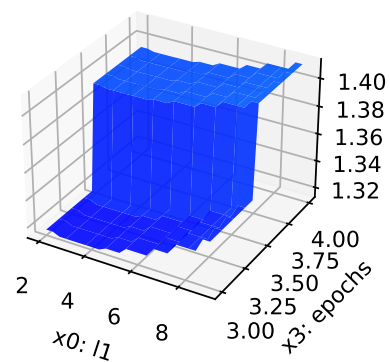
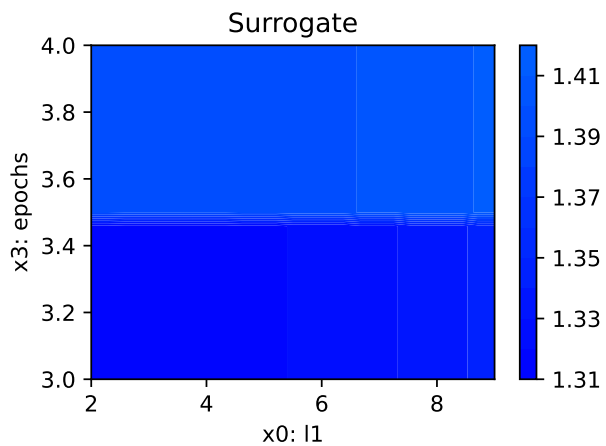
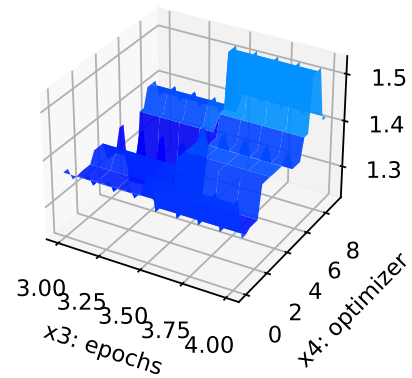
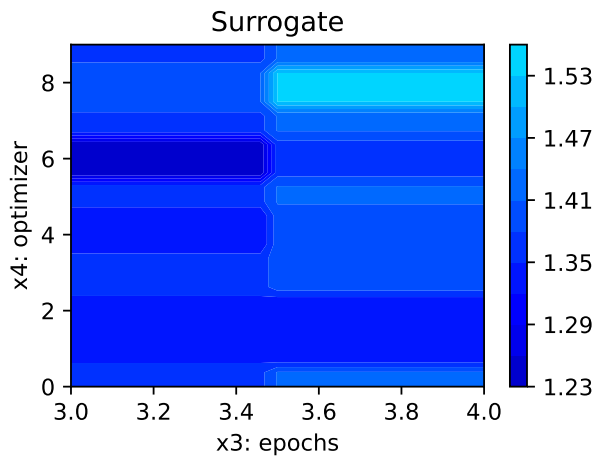
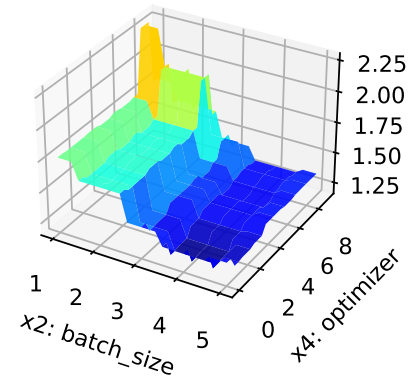
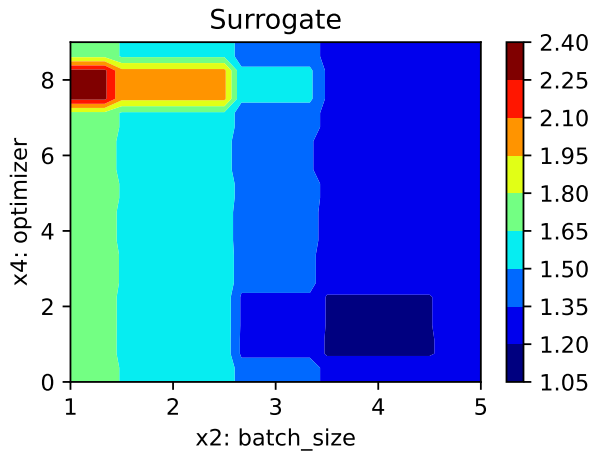


Figure 14.5: Contour plots.





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	0.2.50
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_2023-06-28_04-08-23

```

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: initi_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: -0.3459119496855346 [-----] 3.29%

spotPython tuning: -0.35597484276729563 [#-----] 10.02%

spotPython tuning: -0.3622641509433962 [##-----] 16.24%

spotPython tuning: -0.3622641509433962 [##-----] 22.86%

spotPython tuning: -0.3622641509433962 [###-----] 29.64%

spotPython tuning: -0.3622641509433962 [###-----] 33.81%

spotPython tuning: -0.3622641509433962 [####-----] 39.19%

spotPython tuning: -0.3622641509433962 [####-----] 44.83%

spotPython tuning: -0.3622641509433962 [#####-----] 48.11%

spotPython tuning: -0.3622641509433962 [#####-----] 53.47%

spotPython tuning: -0.3622641509433962 [#####-----] 58.61%

spotPython tuning: -0.3622641509433962 [#####-----] 63.81%

spotPython tuning: -0.3622641509433962 [#####-----] 70.92%

spotPython tuning: -0.3622641509433962 [#####-----] 77.39%

spotPython tuning: -0.3622641509433962 [#####-----] 84.12%

```

```
spotPython tuning: -0.3622641509433962 [#####-] 90.58%

spotPython tuning: -0.3622641509433962 [#####] 97.34%

spotPython tuning: -0.3622641509433962 [#####] 100.00% Done...

<spotPython.spot.spot.Spot at 0x1880f7670>
```

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")
```

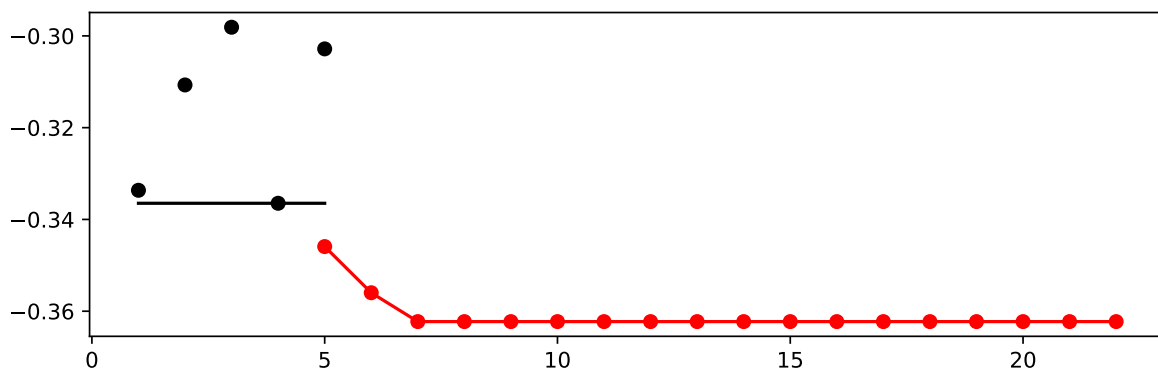


Figure 15.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- Print the results


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

name	type	default	lower	upper	tuned	transform
n_estimators	int	7	5.0	10.0	10.0	transform_power
criterion	factor	gini	0.0	2.0	1.0	None
max_depth	int	10	1.0	20.0	15.0	transform_power
min_samples_split	int	2	2.0	100.0	15.0	None
min_samples_leaf	int	1	1.0	25.0	1.0	None
min_weight_fraction_leaf	float	0.0	0.0	0.01	0.01	None
max_features	factor	sqrt	0.0	1.0	0.0	transform_nor
max_leaf_nodes	int	10	7.0	12.0	10.0	transform_power
min_impurity_decrease	float	0.0	0.0	0.01	0.0	None
bootstrap	factor	1	1.0	1.0	1.0	None
oob_score	factor	0	1.0	1.0	1.0	None

15.10.1 Show variable importance

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

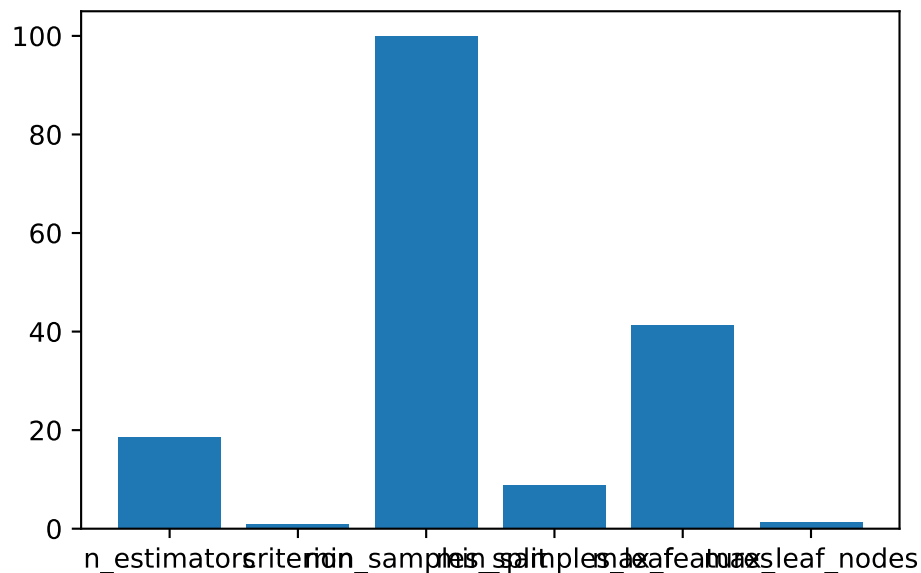


Figure 15.2: Variable importance plot, threshold 0.025.

15.10.2 Get Default Hyperparameters

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

```
[[1.0e+01 1.0e+00 1.5e+01 1.5e+01 1.0e+00 1.0e-02 0.0e+00 1.0e+01 0.0e+00
 1.0e+00 1.0e+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': 1024,
  'criterion': 'entropy',
  'max_depth': 32768,
  'min_samples_split': 15,
  'min_samples_leaf': 1,
  'min_weight_fraction_leaf': 0.01,
  'max_features': 'sqrt',
  'max_leaf_nodes': 1024,
  'min_impurity_decrease': 0.0,
  '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_samples_split=15,
                        min_weight_fraction_leaf=0.01, n_estimators=1024,
                        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

```

0.3502824858757062

```

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: 0.3576271186440678
std_res: 0.006041060238171535
min_res: 0.3436911487758945
max_res: 0.3672316384180791
median_res: 0.3587570621468927

```

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)
```

```
0.34651600753295664
```

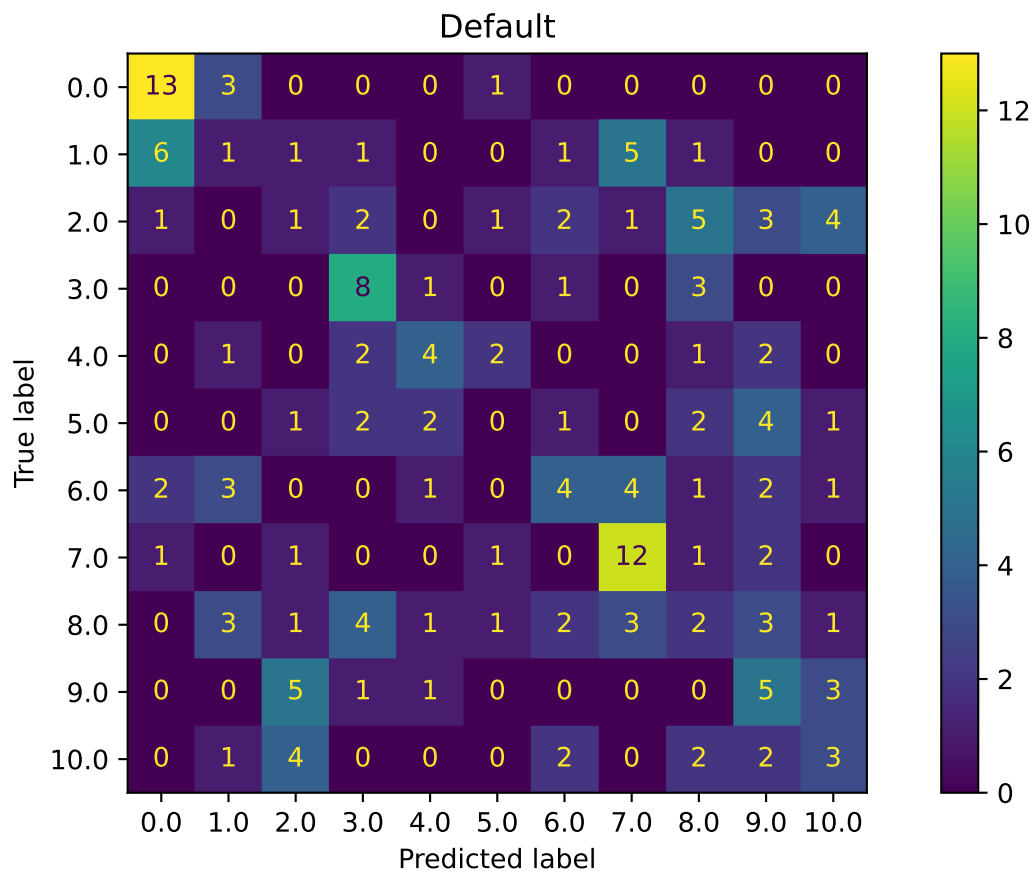
Since one single evaluation is not meaningful, we perform, similar to the evaluation of the SPOT results, $n = 30$ runs of the default setting and calculate the mean and standard deviation of the performance metric.

```
_ = repeated_eval(30, model_default)
```

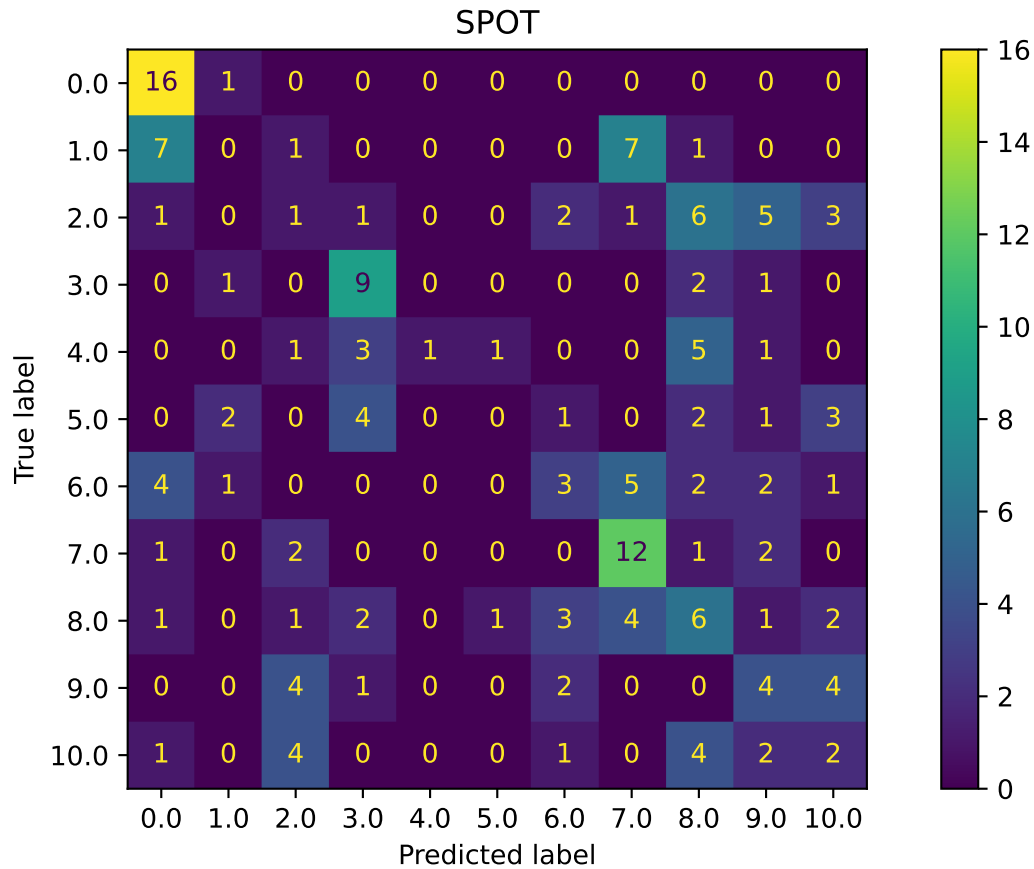
```
mean_res: 0.3444130571249215  
std_res: 0.017007786099273926  
min_res: 0.3032015065913371  
max_res: 0.3747645951035781  
median_res: 0.3451035781544256
```

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)
```

```
(-0.3622641509433962, -0.2981132075471698)
```

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)
```

```
(0.3591194968553459, None)
```

```

fun_control.update({
    "eval": "test_cv",
    "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)

```

(0.3216230936819172, 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)

```

(0.3622334004024145, None)

15.10.9 Detailed Hyperparameter Plots

```

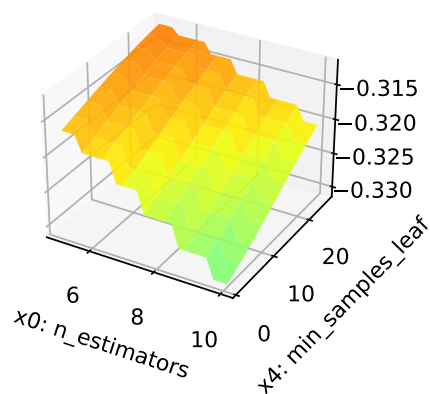
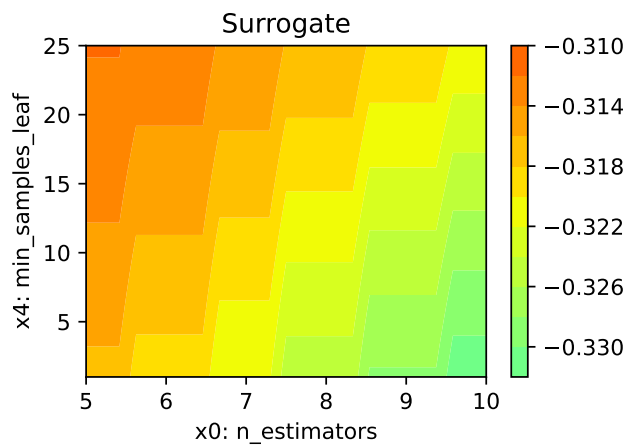
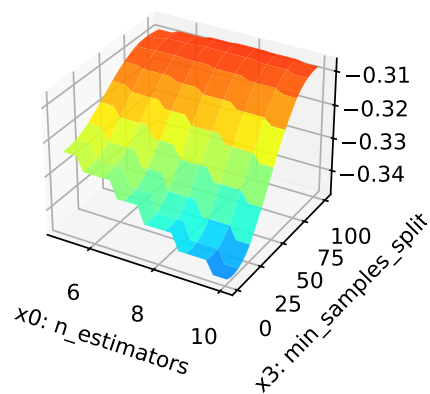
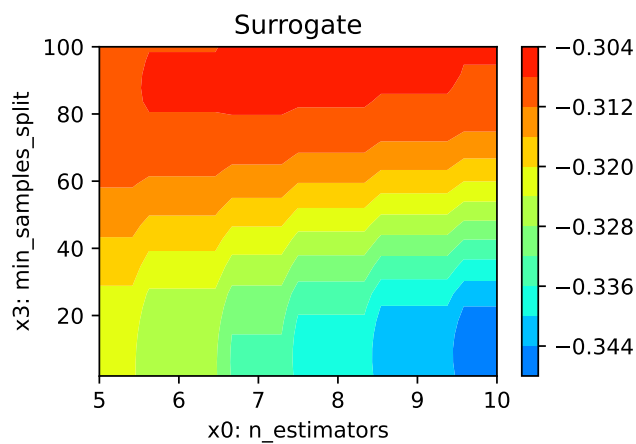
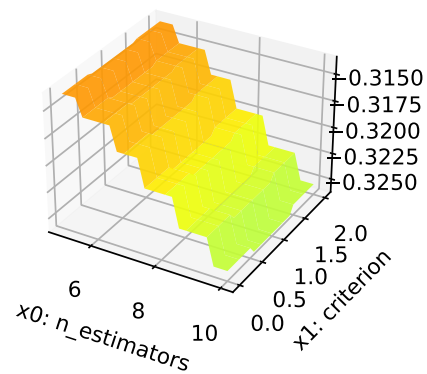
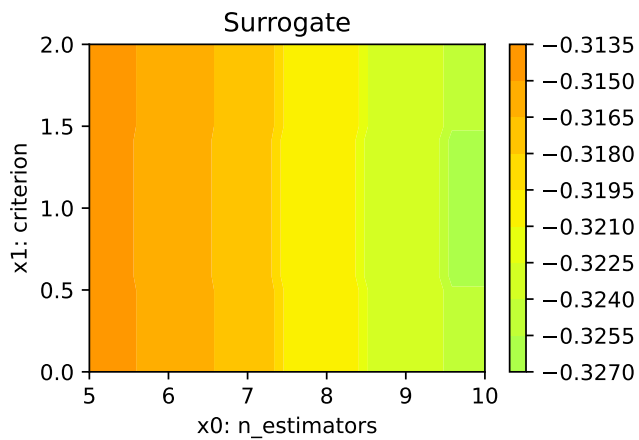
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)

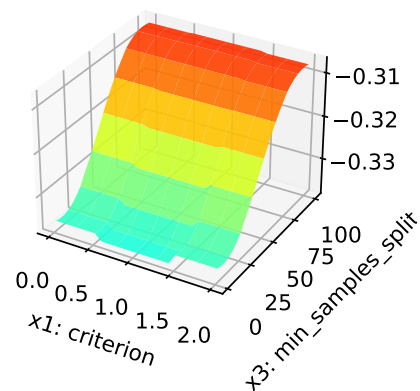
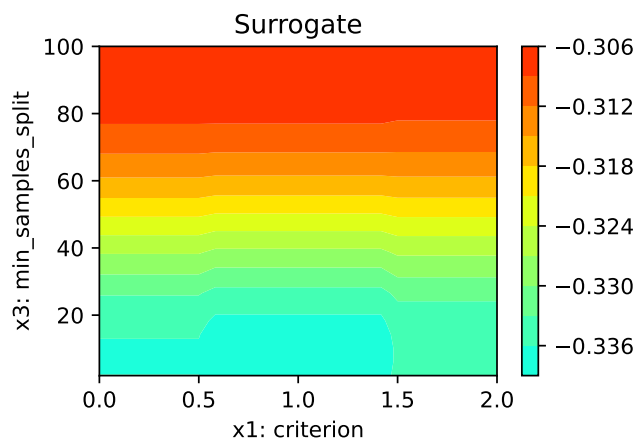
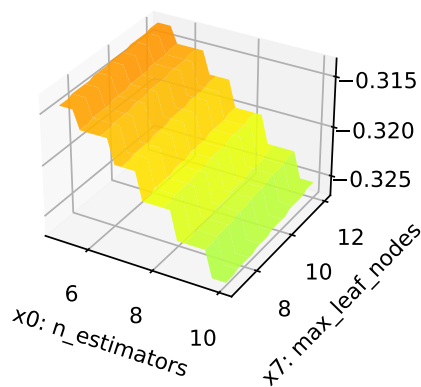
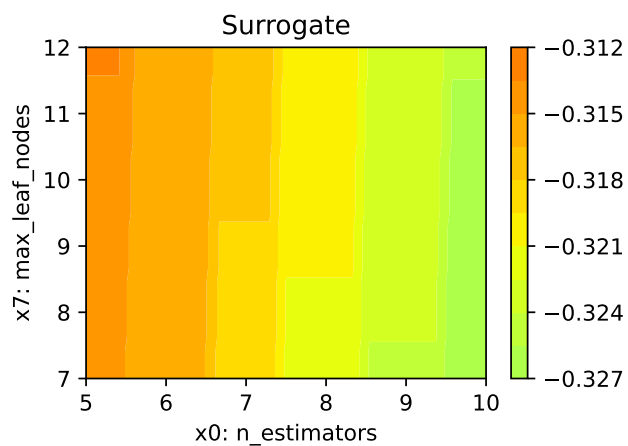
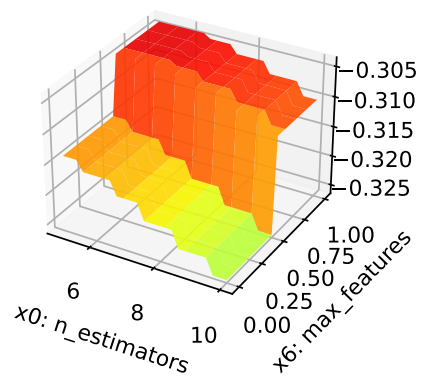
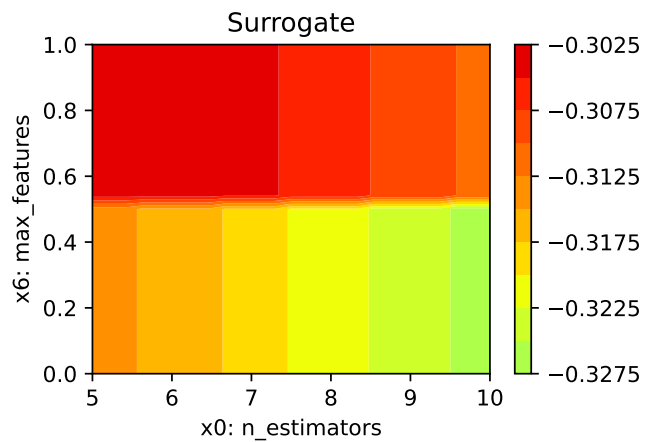
```

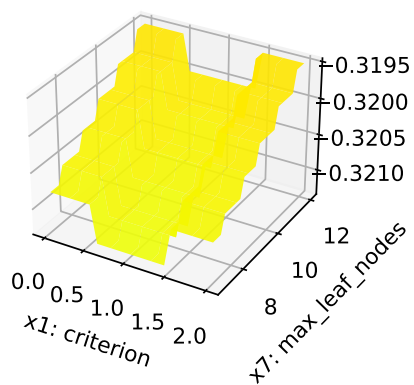
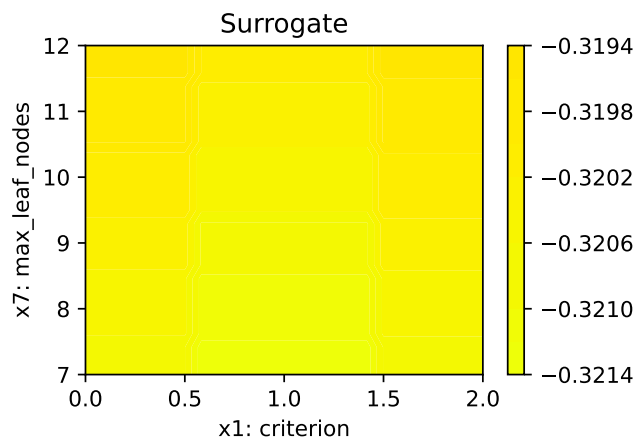
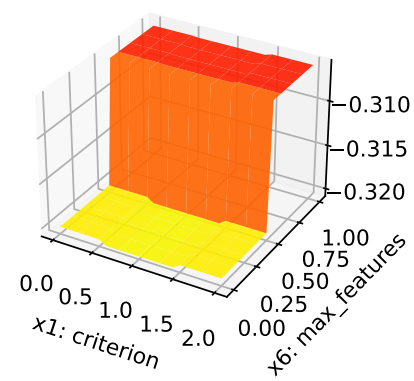
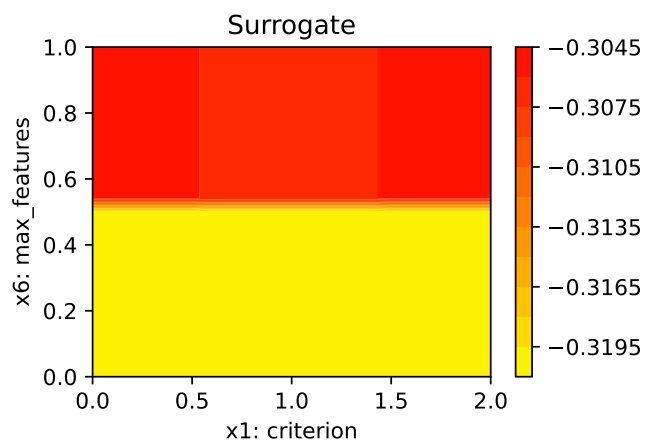
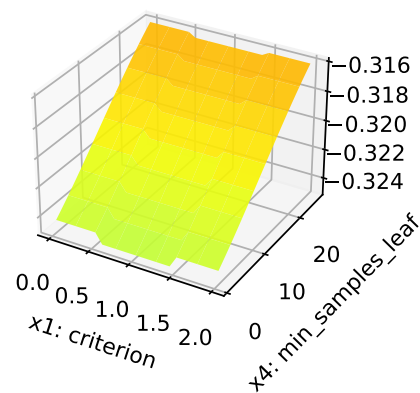
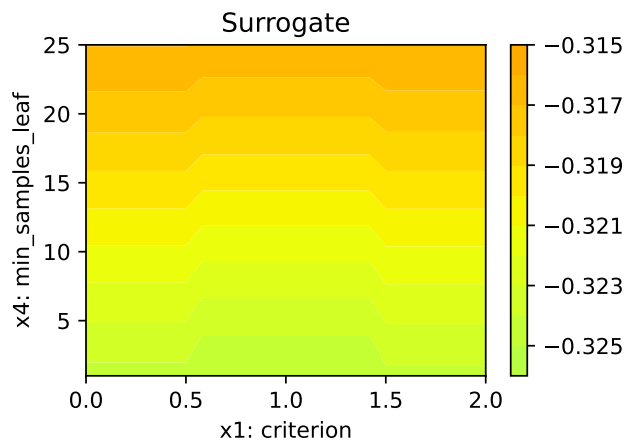
```

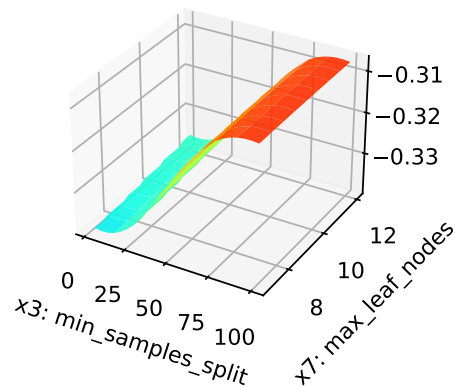
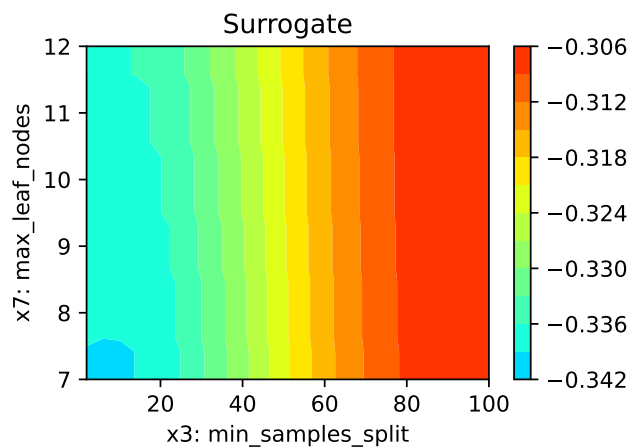
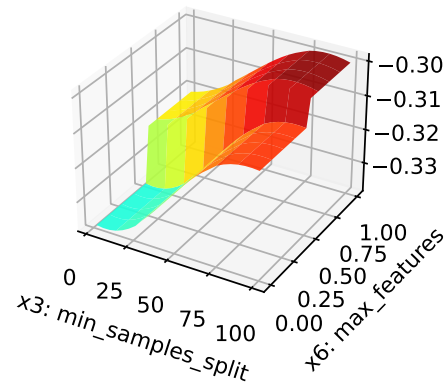
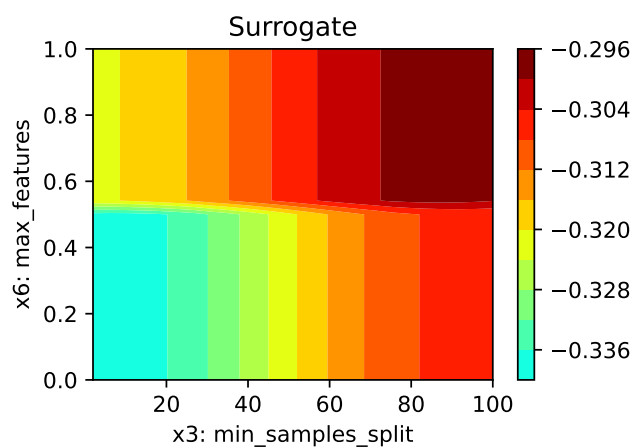
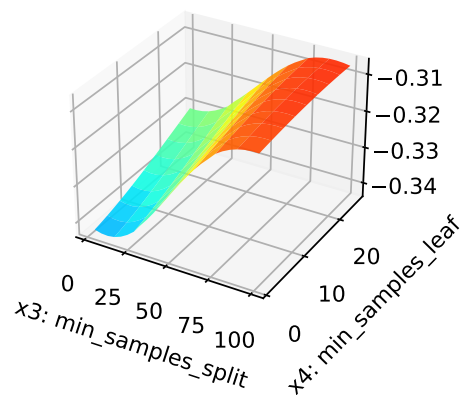
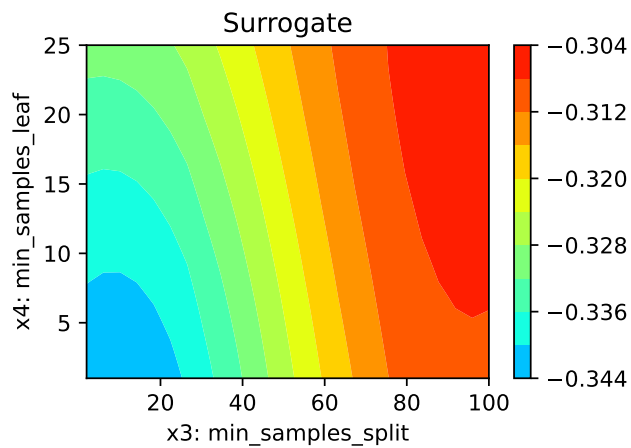
n_estimators: 18.605272172713182
criterion: 0.9915001524980976
min_samples_split: 100.00000000000001
min_samples_leaf: 8.729166846171966
max_features: 41.36872105020423
max_leaf_nodes: 1.358876791825271

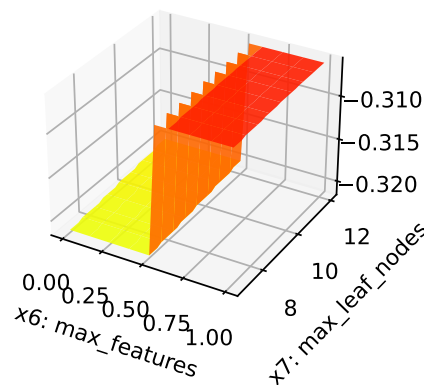
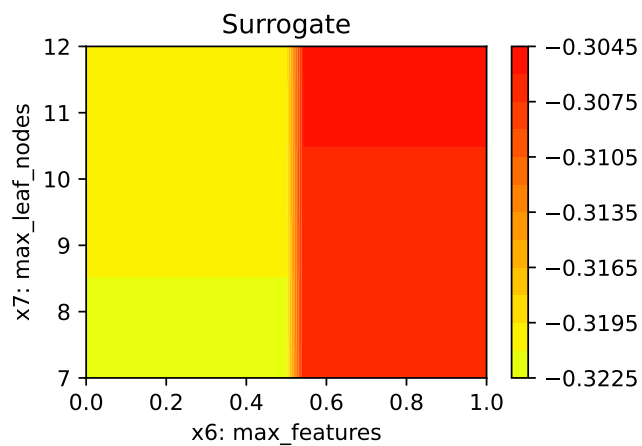
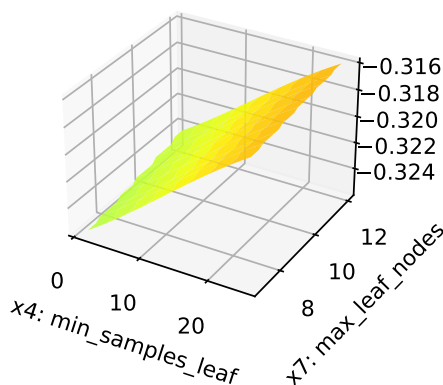
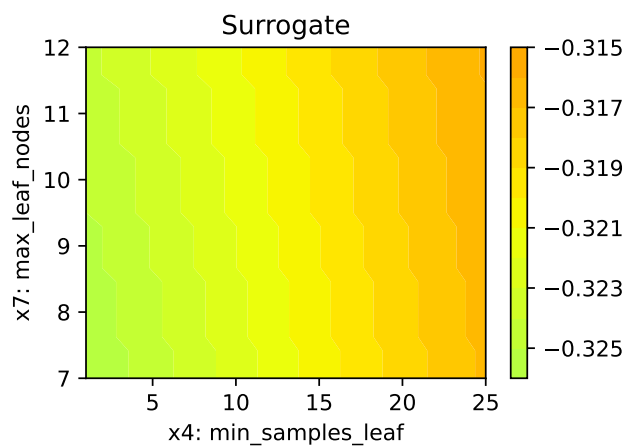
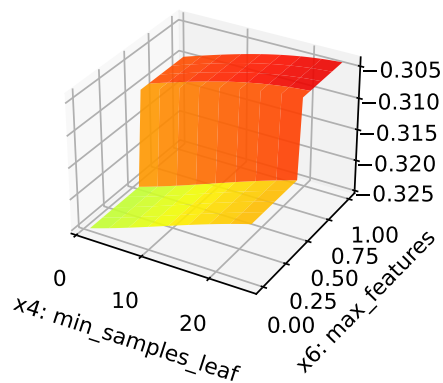
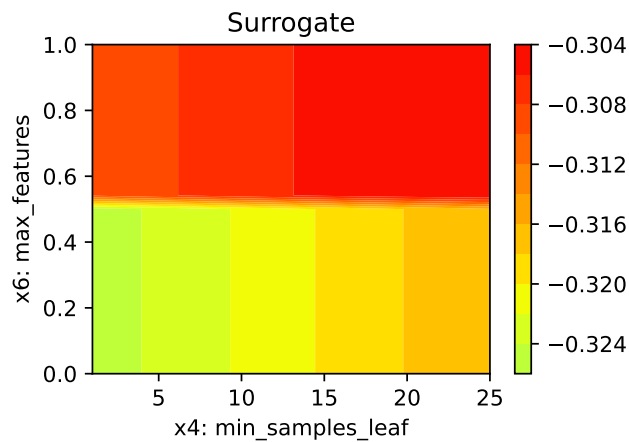
```









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	0.2.50
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_2023-06-28_04-13-43

```

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: -0.36591478696741847 [-----] 4.41%

spotPython tuning: -0.36591478696741847 [#-----] 11.25%

spotPython tuning: -0.36591478696741847 [##-----] 16.43%

spotPython tuning: -0.3659147869674186 [##-----] 22.72%

spotPython tuning: -0.3659147869674186 [###-----] 28.11%

```

spotPython tuning: -0.3659147869674186 [####-----] 35.27%

spotPython tuning: -0.3659147869674186 [####-----] 42.61%

spotPython tuning: -0.3659147869674186 [#####-----] 49.02%

spotPython tuning: -0.3659147869674186 [#####----] 56.45%

spotPython tuning: -0.3771929824561404 [#####----] 64.99%

spotPython tuning: -0.3771929824561404 [#####---] 70.57%

spotPython tuning: -0.3771929824561404 [#####--] 75.52%

spotPython tuning: -0.3771929824561404 [#####-] 93.03%

spotPython tuning: -0.3771929824561404 [#####] 99.73%

spotPython tuning: -0.3771929824561404 [#####] 100.00% Done...

<spotPython.spot.spot.Spot at 0x18825fa90>

```

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")
```

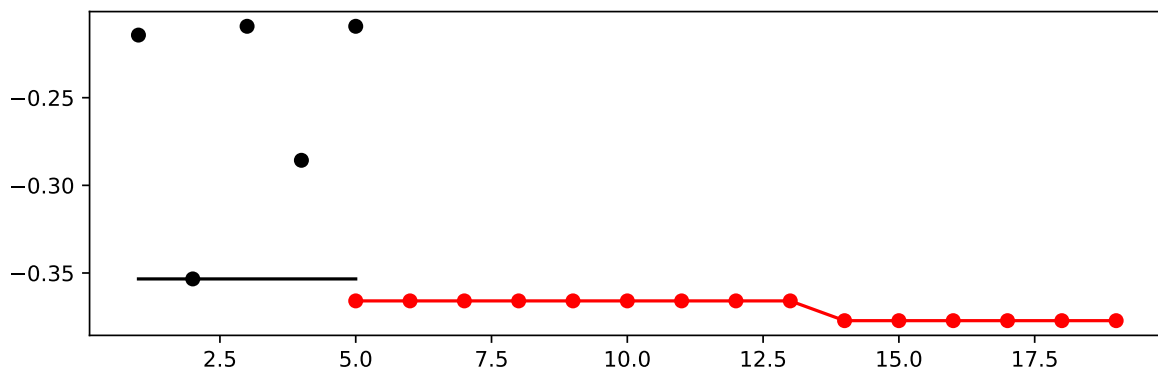


Figure 16.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- 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	-0.8617353576785984	trans
max_iter	int	7	3.0	10.0	7.0	trans
max_leaf_nodes	int	5	1.0	12.0	2.0	trans
max_depth	int	2	1.0	20.0	15.0	trans
min_samples_leaf	int	4	2.0	10.0	4.0	trans
l2_regularization	float	0.0	0.0	10.0	9.345539601378421	None
max_bins	int	255	127.0	255.0	178.0	None
early_stopping	factor	1	0.0	1.0	1.0	None
n_iter_no_change	int	10	5.0	20.0	18.0	None
tol	float	0.0001	1e-05	0.001	0.0009698148152696297	None

16.10.1 Show variable importance

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

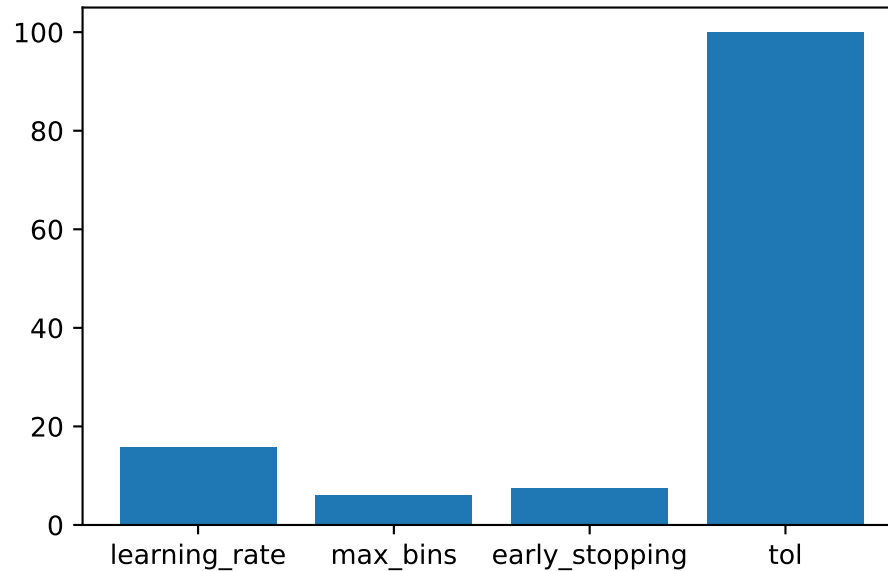


Figure 16.2: Variable importance plot, threshold 0.025.

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 -8.61735358e-01  7.00000000e+00  2.00000000e+00
   1.50000000e+01  4.00000000e+00  9.34553960e+00  1.78000000e+02
   1.00000000e+00  1.80000000e+01  9.69814815e-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': 0.13748795183997445,
 'max_iter': 128,
 'max_leaf_nodes': 4,
 'max_depth': 32768,
 'min_samples_leaf': 16,
 'l2_regularization': 9.345539601378421,
 'max_bins': 178,
 'early_stopping': 1,
 'n_iter_no_change': 18,
 'tol': 0.0009698148152696297}]
```

```

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=9.345539601378421,
                                learning_rate=0.13748795183997445, max_bins=178,
                                max_depth=32768, max_iter=128, max_leaf_nodes=4,
                                min_samples_leaf=16, n_iter_no_change=18,
                                tol=0.0009698148152696297)

```

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

```

```
0.34557438794726936
```

```

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: 0.33763339610797233
std_res: 0.012460449601550256
min_res: 0.3126177024482109
max_res: 0.36911487758945394
median_res: 0.3375706214689265

```

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)

```

```
0.3625235404896422
```

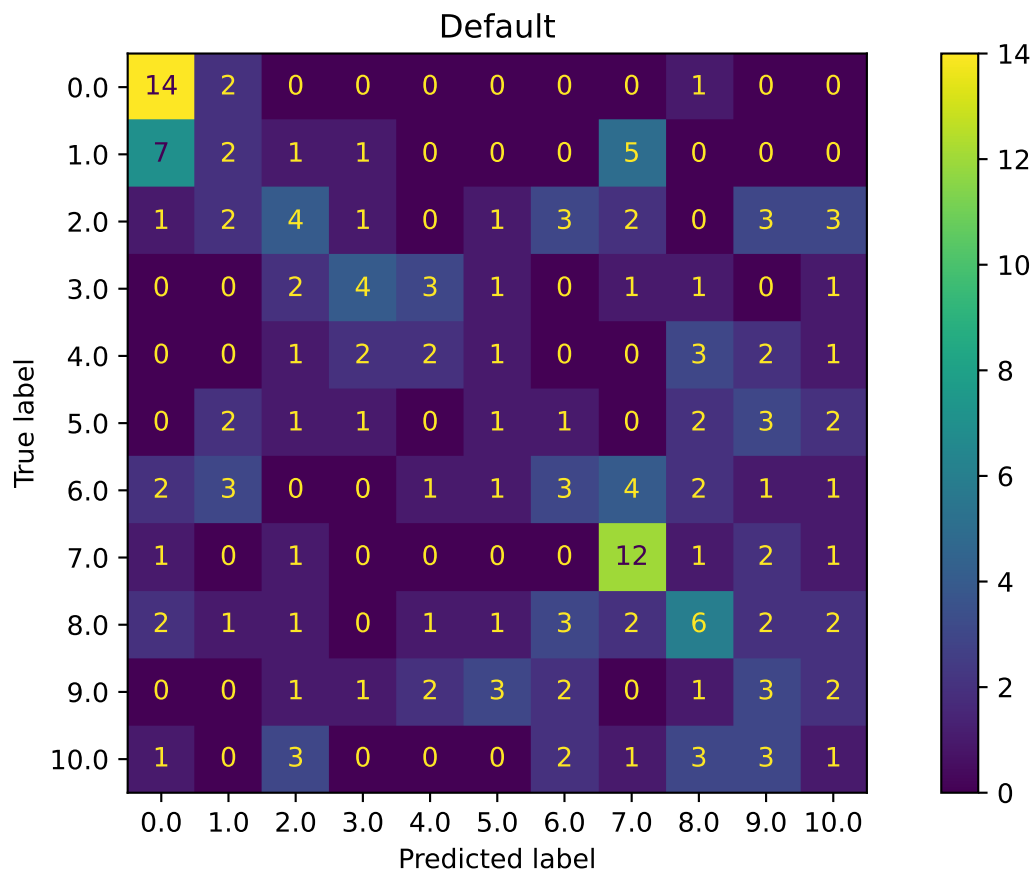

Since one single evaluation is not meaningful, we perform, similar to the evaluation of the SPOT results, $n = 30$ runs of the default setting and calculate the mean and standard deviation of the performance metric.

```
_ = repeated_eval(30, model_default)
```

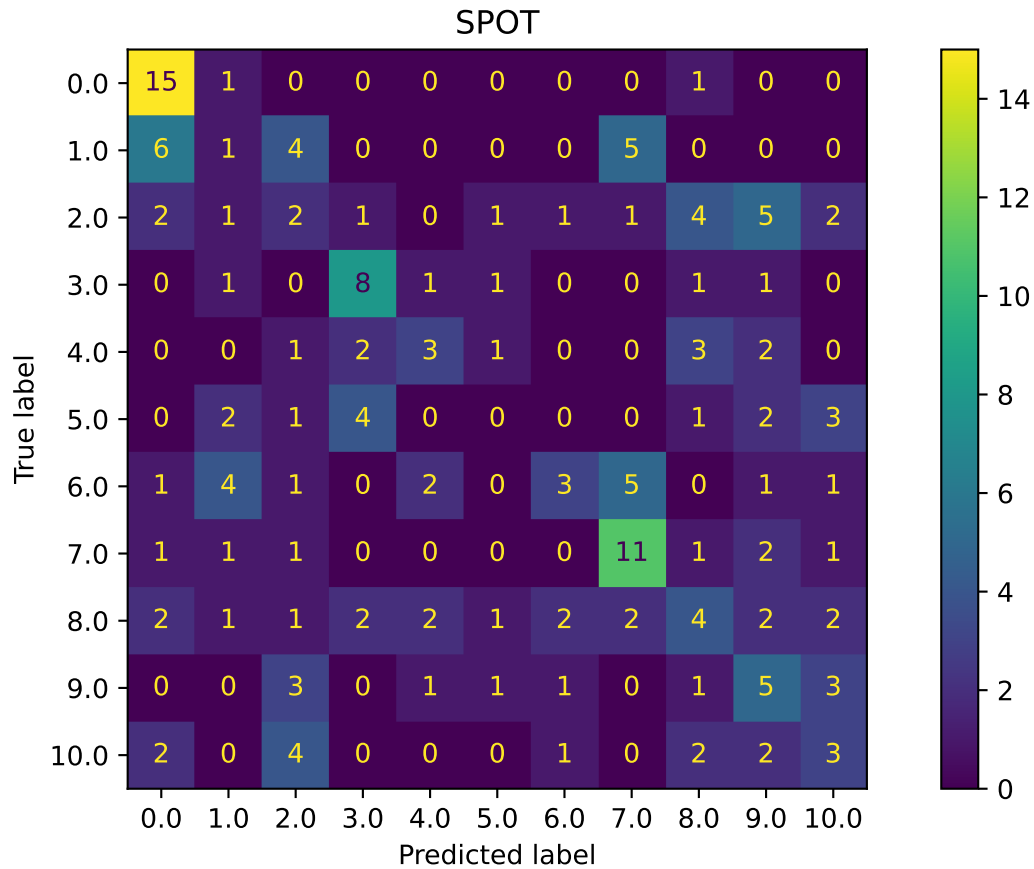
```
mean_res: 0.3432203389830509
std_res: 0.013098895293132257
min_res: 0.3182674199623352
max_res: 0.3672316384180791
median_res: 0.34133709981167604
```

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)
```

```
(-0.3771929824561404, -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)
```

```
(0.3433962264150944, None)
```

```

fun_control.update({
    "eval": "test_cv",
    "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)

```

(0.2898692810457516, 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)

```

(0.35940308517773306, None)

16.10.9 Detailed Hyperparameter Plots

```

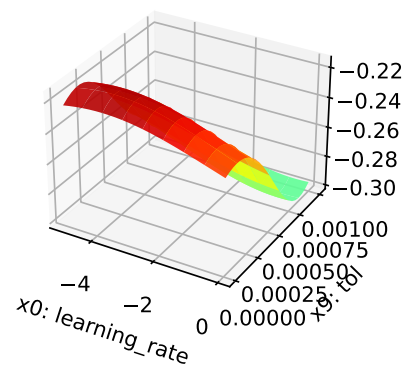
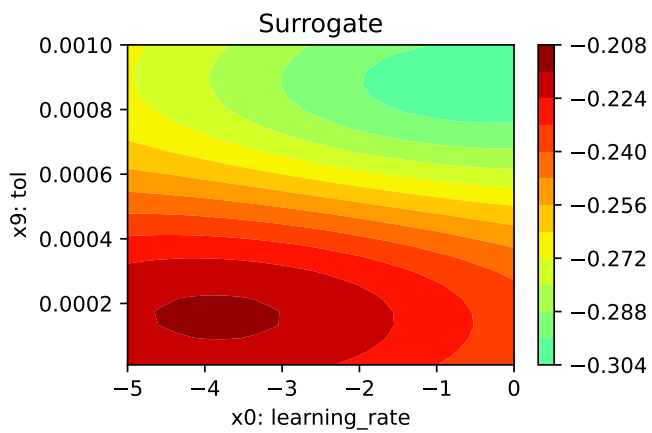
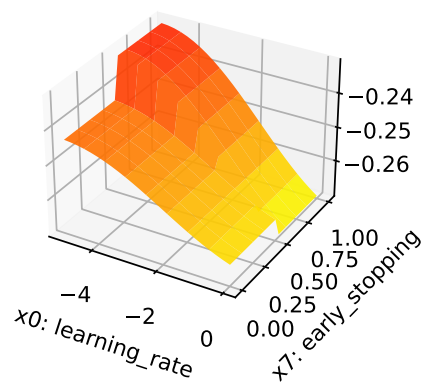
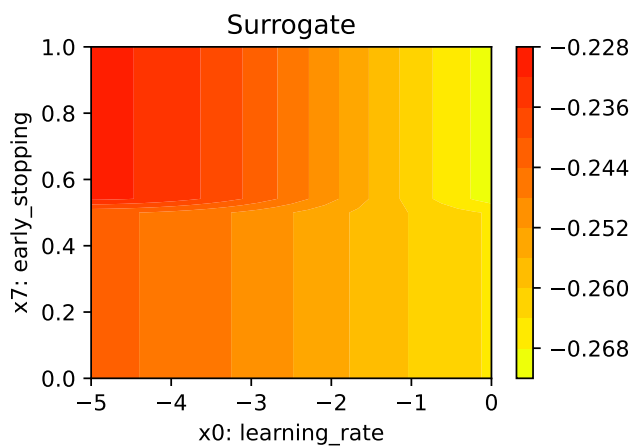
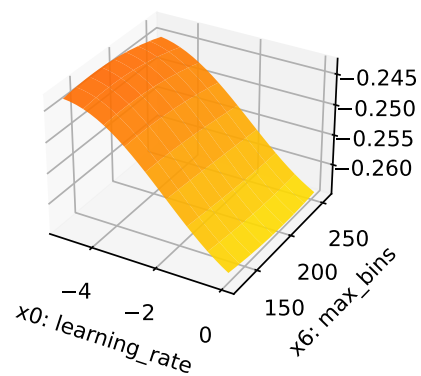
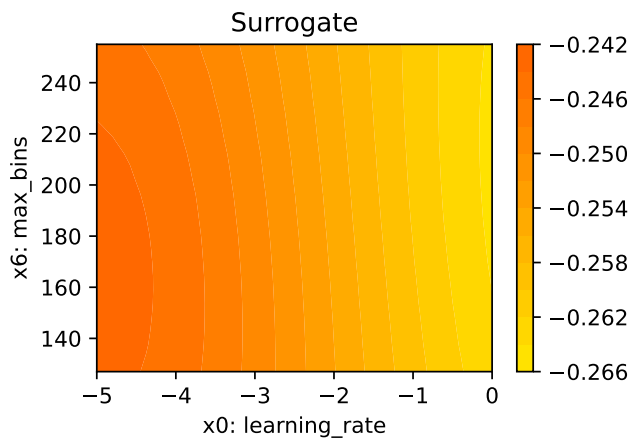
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)

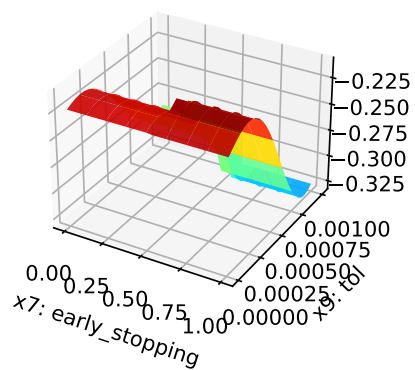
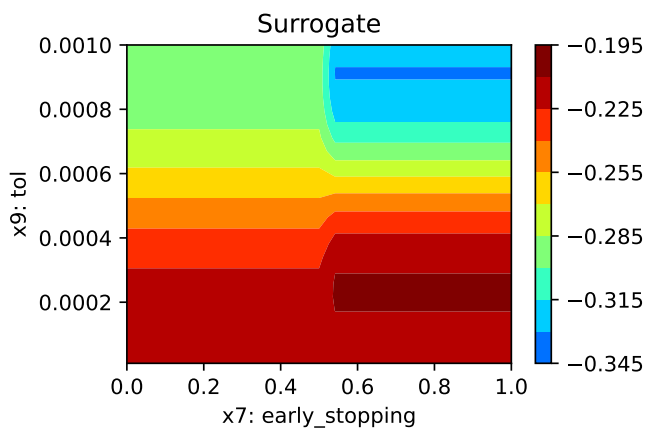
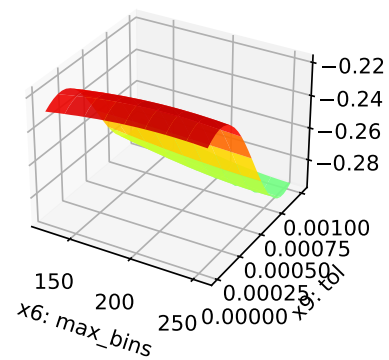
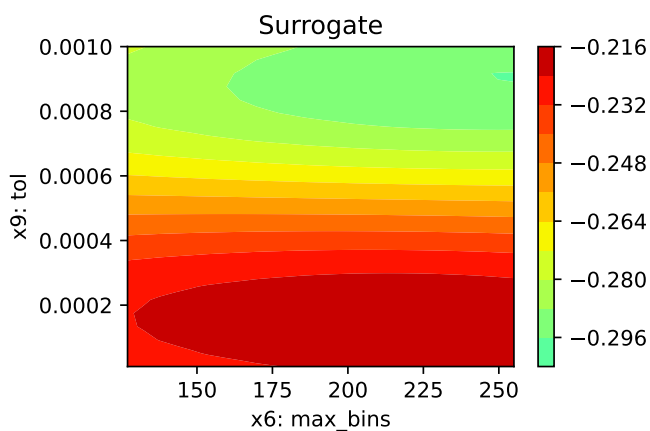
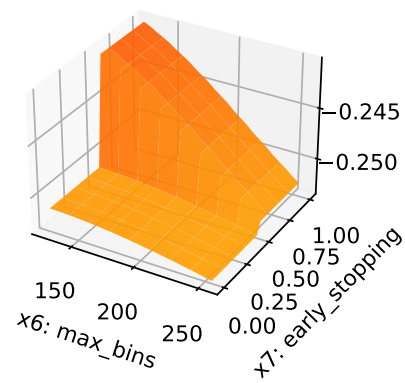
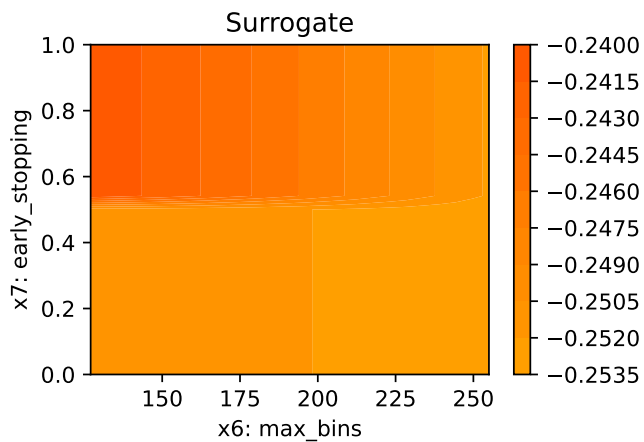
```

```

learning_rate: 15.80867960227496
max_bins: 6.054395073249785
early_stopping: 7.417698113607669
tol: 100.0

```





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	0.2.50
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_2023-06-28_04-17-41

```

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: -0.3897243107769423 [-----] 0.82%

spotPython tuning: -0.3897243107769423 [-----] 1.94%

spotPython tuning: -0.3897243107769423 [-----] 2.84%

spotPython tuning: -0.3897243107769423 [-----] 3.70%

spotPython tuning: -0.3897243107769423 [-----] 4.54%

spotPython tuning: -0.3897243107769423 [#-----] 5.23%

spotPython tuning: -0.3897243107769423 [#-----] 5.91%

spotPython tuning: -0.3897243107769423 [#-----] 6.62%

spotPython tuning: -0.3897243107769423 [#-----] 7.34%

spotPython tuning: -0.3897243107769423 [#-----] 8.13%

spotPython tuning: -0.3897243107769423 [#-----] 8.87%

spotPython tuning: -0.3897243107769423 [#-----] 9.62%

```

spotPython tuning: -0.3897243107769423 [#-----] 10.63%

spotPython tuning: -0.3897243107769423 [#-----] 11.50%

spotPython tuning: -0.3897243107769423 [#-----] 12.49%

spotPython tuning: -0.3897243107769423 [#-----] 13.61%

spotPython tuning: -0.3897243107769423 [##-----] 15.13%

spotPython tuning: -0.3897243107769423 [##-----] 16.13%

spotPython tuning: -0.3897243107769423 [##-----] 17.13%

spotPython tuning: -0.3897243107769423 [##-----] 18.21%

spotPython tuning: -0.3897243107769423 [##-----] 19.36%

spotPython tuning: -0.3897243107769423 [##-----] 20.36%

spotPython tuning: -0.3897243107769423 [##-----] 21.38%

spotPython tuning: -0.3897243107769423 [##-----] 22.64%

spotPython tuning: -0.3897243107769423 [##-----] 23.74%

spotPython tuning: -0.3897243107769423 [##-----] 24.85%

spotPython tuning: -0.3897243107769423 [###-----] 25.84%

spotPython tuning: -0.3897243107769423 [###-----] 27.09%

spotPython tuning: -0.3897243107769423 [###-----] 28.22%

spotPython tuning: -0.3897243107769423 [###-----] 29.24%

spotPython tuning: -0.3897243107769423 [###-----] 30.33%

spotPython tuning: -0.3897243107769423 [###-----] 31.37%

spotPython tuning: -0.3897243107769423 [###-----] 32.59%

spotPython tuning: -0.3897243107769423 [###-----] 33.68%

spotPython tuning: -0.3897243107769423 [###-----] 34.76%

spotPython tuning: -0.3897243107769423 [####-----] 35.85%

spotPython tuning: -0.3897243107769423 [####-----] 37.23%

spotPython tuning: -0.3897243107769423 [####-----] 38.45%

spotPython tuning: -0.3897243107769423 [####-----] 39.58%

spotPython tuning: -0.3897243107769423 [####-----] 41.01%

spotPython tuning: -0.3897243107769423 [####-----] 42.21%

spotPython tuning: -0.3897243107769423 [####-----] 43.22%

spotPython tuning: -0.3897243107769423 [####-----] 44.52%

spotPython tuning: -0.3897243107769423 [#####-----] 45.72%

spotPython tuning: -0.3897243107769423 [#####-----] 46.90%

spotPython tuning: -0.3897243107769423 [#####-----] 48.33%

spotPython tuning: -0.3897243107769423 [#####-----] 49.98%

spotPython tuning: -0.3897243107769423 [#####-----] 51.22%

spotPython tuning: -0.3897243107769423 [#####-----] 52.57%

spotPython tuning: -0.3897243107769423 [#####-----] 53.87%

spotPython tuning: -0.3897243107769423 [#####----] 55.29%

spotPython tuning: -0.3897243107769423 [#####----] 56.80%

spotPython tuning: -0.3897243107769423 [#####----] 58.45%

spotPython tuning: -0.3897243107769423 [#####----] 60.04%

spotPython tuning: -0.3897243107769423 [#####----] 61.69%

spotPython tuning: -0.3897243107769423 [#####----] 63.21%

spotPython tuning: -0.3897243107769423 [#####----] 65.17%

spotPython tuning: -0.3897243107769423 [#####---] 66.46%

spotPython tuning: -0.3897243107769423 [#####---] 68.45%

spotPython tuning: -0.3897243107769423 [#####---] 70.12%

spotPython tuning: -0.3897243107769423 [#####---] 72.31%

spotPython tuning: -0.3897243107769423 [#####---] 74.95%

spotPython tuning: -0.3897243107769423 [#####--] 77.70%

spotPython tuning: -0.3897243107769423 [#####--] 79.57%

spotPython tuning: -0.3897243107769423 [#####--] 81.22%

spotPython tuning: -0.3897243107769423 [#####--] 83.26%

spotPython tuning: -0.3897243107769423 [#####-] 85.50%

spotPython tuning: -0.3897243107769423 [#####-] 87.48%

spotPython tuning: -0.3897243107769423 [#####-] 89.20%

```
spotPython tuning: -0.3897243107769423 [#####-] 90.96%

spotPython tuning: -0.3897243107769423 [#####-] 92.83%

spotPython tuning: -0.3897243107769423 [#####-] 94.84%

spotPython tuning: -0.3897243107769423 [#####] 96.47%

spotPython tuning: -0.3897243107769423 [#####] 98.32%

spotPython tuning: -0.3897243107769423 [#####] 100.00% Done...

<spotPython.spot.spot.Spot at 0x17bd5f640>
```

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")
```

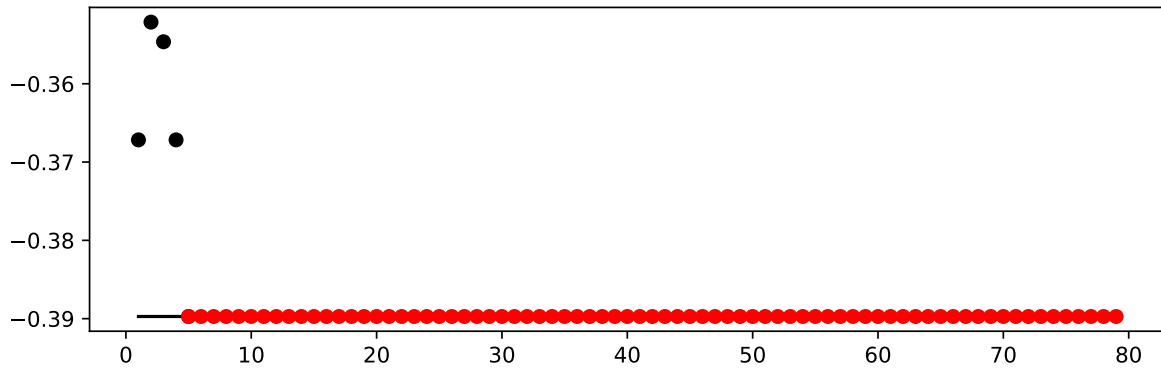


Figure 17.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- 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	7.65501078489161	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	0.0	None
probability	factor	0	1.0	1.0	1.0	None
tol	float	0.001	0.0001	0.01	0.006578078210197142	None
cache_size	float	200.0	100.0	400.0	224.8839704935874	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
```

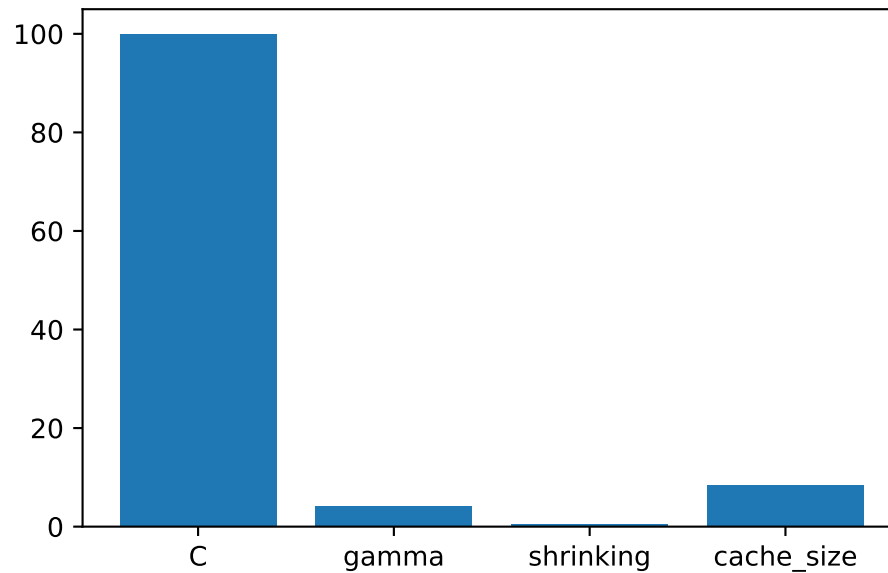



Figure 17.2: Variable importance plot, threshold 0.025.

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)
```

```
[[7.65501078e+00 0.00000000e+00 3.00000000e+00 1.00000000e+00
  0.00000000e+00 0.00000000e+00 1.00000000e+00 6.57807821e-03
  2.24883970e+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': 7.65501078489161,
  'kernel': 'rbf',
  'degree': 3,
  'gamma': 'auto',
  'coef0': 0.0,
  'shrinking': 0,
  'probability': 1,
  'tol': 0.006578078210197142,
  'cache_size': 224.8839704935874,
  '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=7.65501078489161, break_ties=0, cache_size=224.8839704935874,
    gamma='auto', probability=1, shrinking=0, tol=0.006578078210197142)

```

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

```

```
0.3559322033898305
```

```

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: 0.3636534839924671
std_res: 0.0052336884738467615
min_res: 0.3512241054613936
max_res: 0.3700564971751412
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)

```

```

0.3926553672316384

```

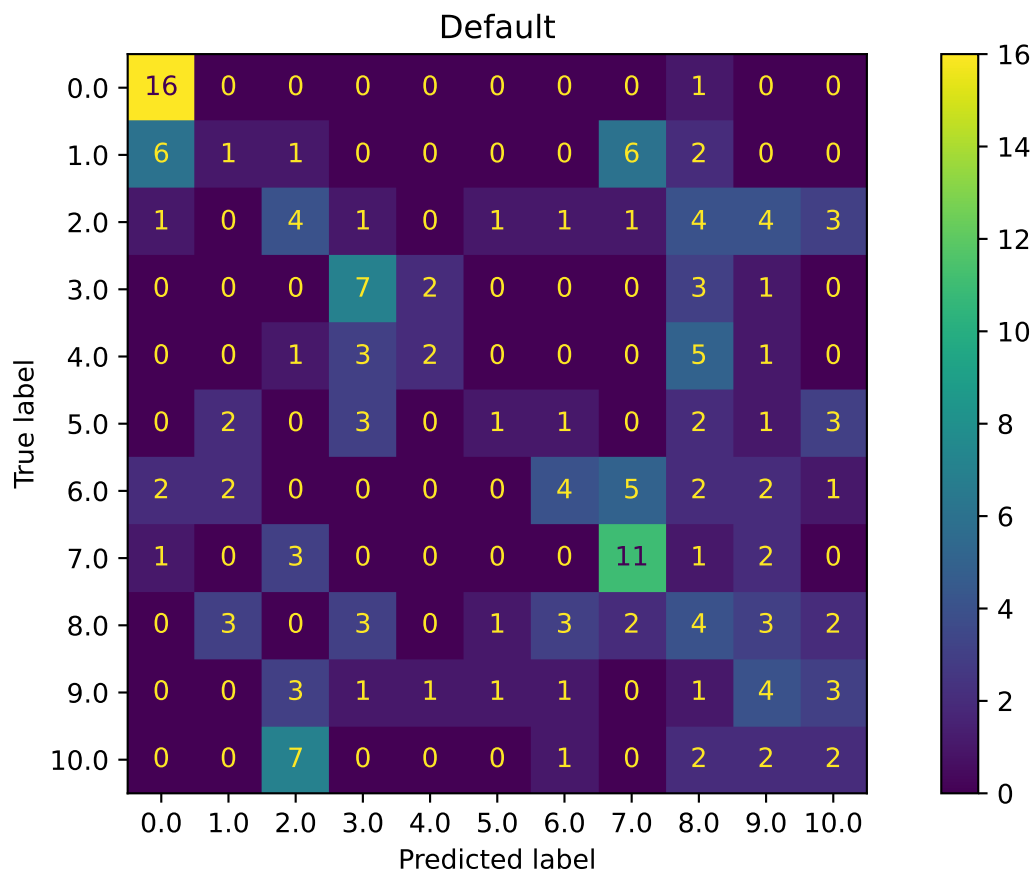
Since one single evaluation is not meaningful, we perform, similar to the evaluation of the SPOT results, $n = 30$ runs of the default setting and calculate the mean and standard deviation of the performance metric.

```
_ = repeated_eval(30, model_default)
```

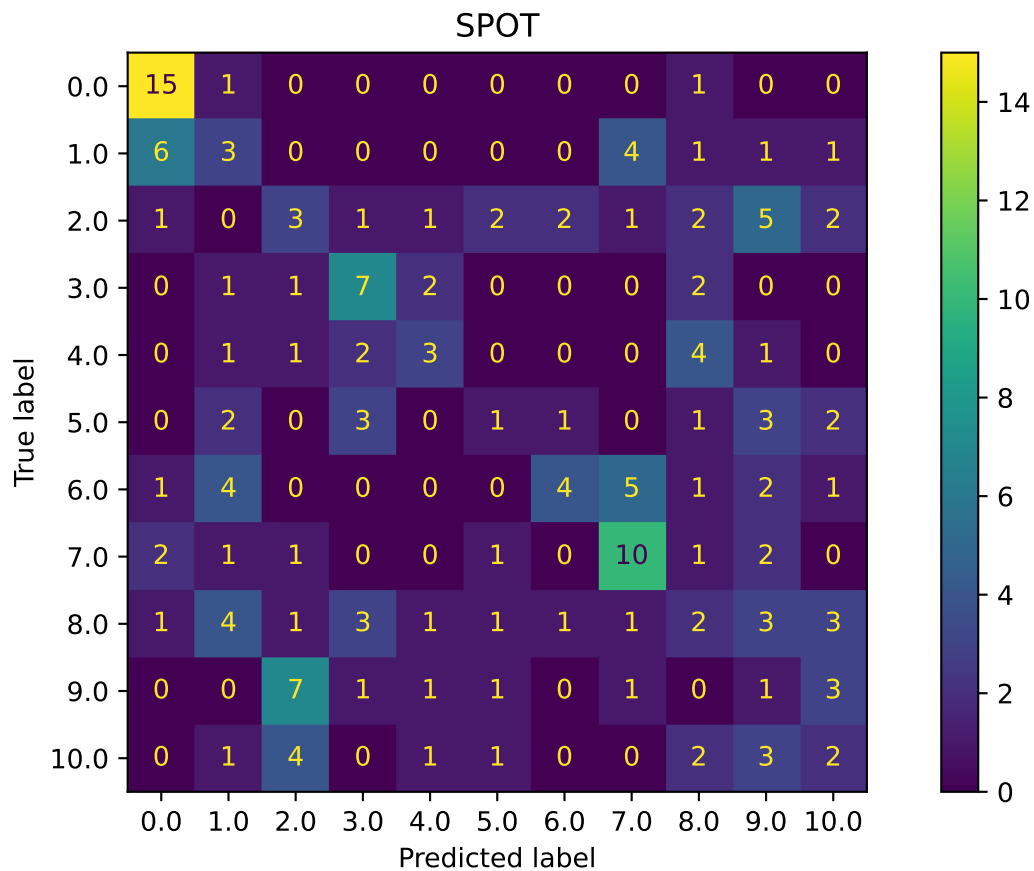
```
mean_res: 0.3845574387947269
std_res: 0.0043259879859435186
min_res: 0.37664783427495285
max_res: 0.3964218455743879
median_res: 0.384180790960452
```

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)
```

```
(-0.3897243107769423, -0.3245614035087719)
```

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)
```

(0.3383647798742138, None)

```
fun_control.update({
    "eval": "test_cv",
    "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)
```

(0.35876906318082785, None)

- This is the evaluation that will be used in the comparison:

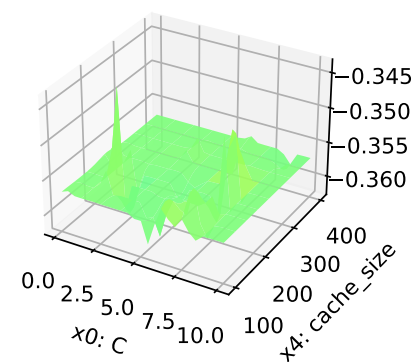
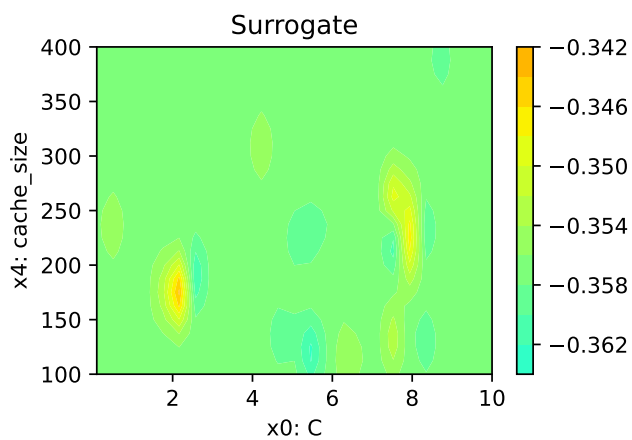
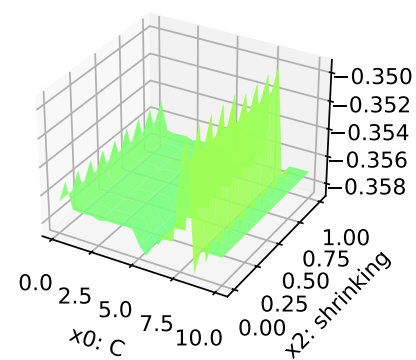
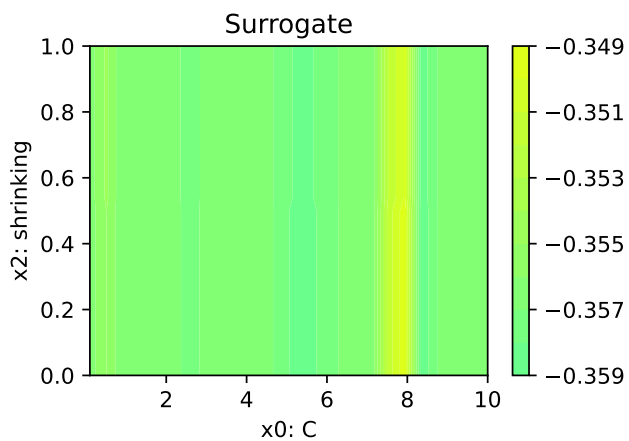
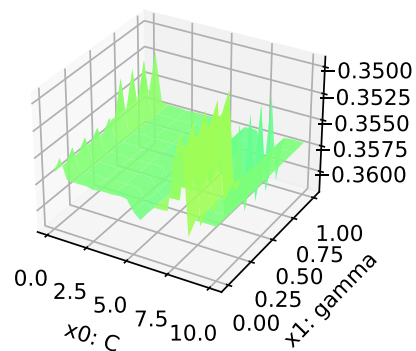
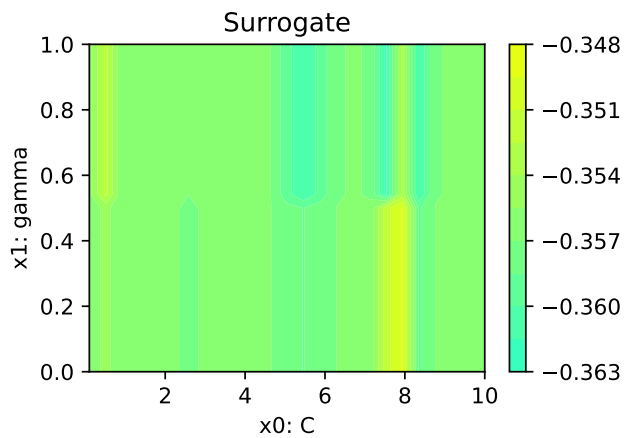
```
fun_control.update({
    "eval": "data_cv",
    "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)
```

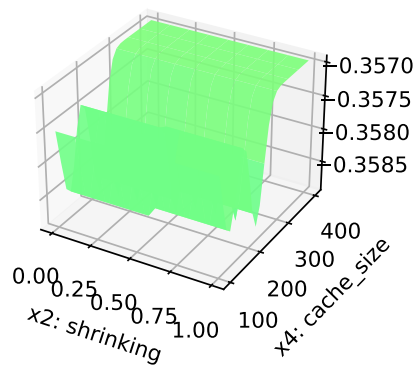
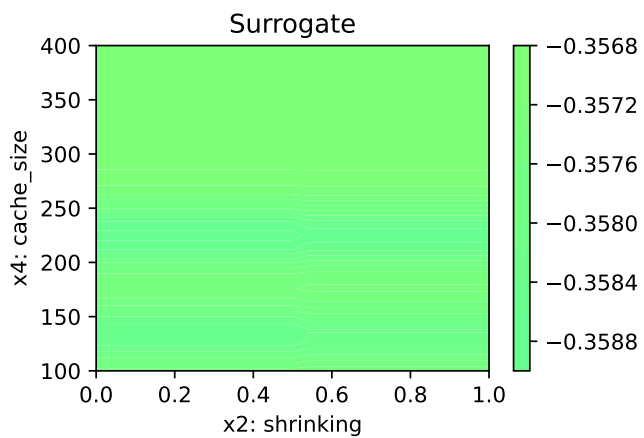
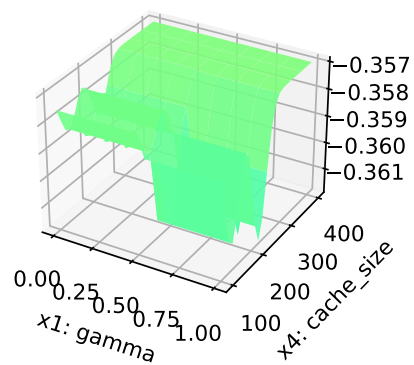
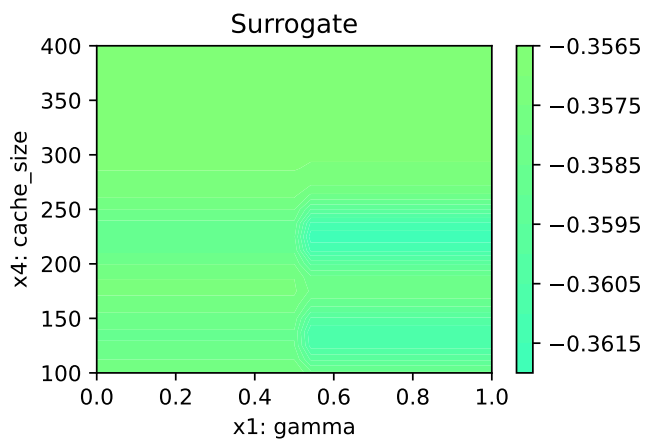
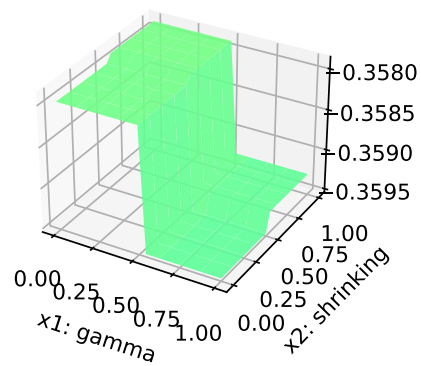
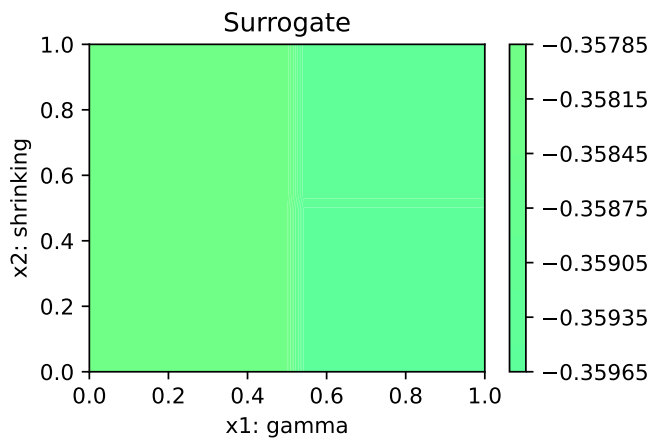
(0.35276995305164316, None)

17.10.9 Detailed Hyperparameter Plots

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

```
C: 100.0
gamma: 4.058673111386977
shrinking: 0.4176575884479082
cache_size: 8.36540842643731
```





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	0.2.50
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_2023-06-28_04-20-42

```

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/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`.

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 [-----] 0.71%

spotPython tuning: -0.3107769423558897 [-----] 1.50%

spotPython tuning: -0.3107769423558897 [-----] 2.27%

spotPython tuning: -0.3107769423558897 [-----] 3.00%

spotPython tuning: -0.3107769423558897 [-----] 3.76%

spotPython tuning: -0.3107769423558897 [-----] 4.66%

spotPython tuning: -0.3107769423558897 [#-----] 5.85%

spotPython tuning: -0.3107769423558897 [#-----] 6.87%

spotPython tuning: -0.3107769423558897 [#-----] 7.78%

spotPython tuning: -0.3107769423558897 [#-----] 8.70%

spotPython tuning: -0.3107769423558897 [#-----] 9.65%

spotPython tuning: -0.3107769423558897 [#-----] 10.93%

spotPython tuning: -0.3107769423558897 [#-----] 12.30%

spotPython tuning: -0.3107769423558897 [#-----] 13.79%

spotPython tuning: -0.3107769423558897 [##-----] 15.24%

spotPython tuning: -0.3107769423558897 [##-----] 16.87%

spotPython tuning: -0.3107769423558897 [##-----] 18.85%

spotPython tuning: -0.3107769423558897 [##-----] 20.29%

spotPython tuning: -0.3107769423558897 [##-----] 22.01%

spotPython tuning: -0.3107769423558897 [##-----] 23.44%

spotPython tuning: -0.3107769423558897 [##-----] 24.83%

spotPython tuning: -0.3107769423558897 [###-----] 26.09%

spotPython tuning: -0.3107769423558897 [###-----] 27.51%

spotPython tuning: -0.3107769423558897 [###-----] 29.25%

spotPython tuning: -0.3107769423558897 [###-----] 30.71%

spotPython tuning: -0.3107769423558897 [###-----] 32.53%

spotPython tuning: -0.3107769423558897 [###-----] 33.96%

spotPython tuning: -0.3107769423558897 [####-----] 35.55%

spotPython tuning: -0.3107769423558897 [####-----] 37.32%

spotPython tuning: -0.3107769423558897 [####-----] 39.03%

spotPython tuning: -0.3107769423558897 [####-----] 40.62%

spotPython tuning: -0.3107769423558897 [####-----] 42.53%

spotPython tuning: -0.3107769423558897 [####-----] 44.06%

spotPython tuning: -0.3107769423558897 [#####-----] 46.10%

spotPython tuning: -0.3107769423558897 [#####-----] 48.37%

spotPython tuning: -0.3107769423558897 [#####-----] 50.38%

spotPython tuning: -0.3107769423558897 [#####-----] 52.36%

spotPython tuning: -0.3107769423558897 [#####-----] 54.65%

spotPython tuning: -0.3107769423558897 [#####-----] 56.90%

spotPython tuning: -0.3107769423558897 [#####-----] 58.94%

spotPython tuning: -0.3107769423558897 [#####-----] 61.28%

spotPython tuning: -0.3107769423558897 [#####-----] 63.53%

spotPython tuning: -0.3107769423558897 [#####-----] 65.64%

```

spotPython tuning: -0.3107769423558897 [#####---] 67.78%

spotPython tuning: -0.3107769423558897 [#####---] 70.13%

spotPython tuning: -0.3107769423558897 [#####---] 73.00%

spotPython tuning: -0.3107769423558897 [#####--] 76.21%

spotPython tuning: -0.3107769423558897 [#####--] 78.93%

spotPython tuning: -0.3107769423558897 [#####--] 81.40%

spotPython tuning: -0.3107769423558897 [#####--] 84.65%

spotPython tuning: -0.3107769423558897 [#####] 100.00% Done...

<spotPython.spot.spot.Spot at 0x1869bfa90>

```

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")

```

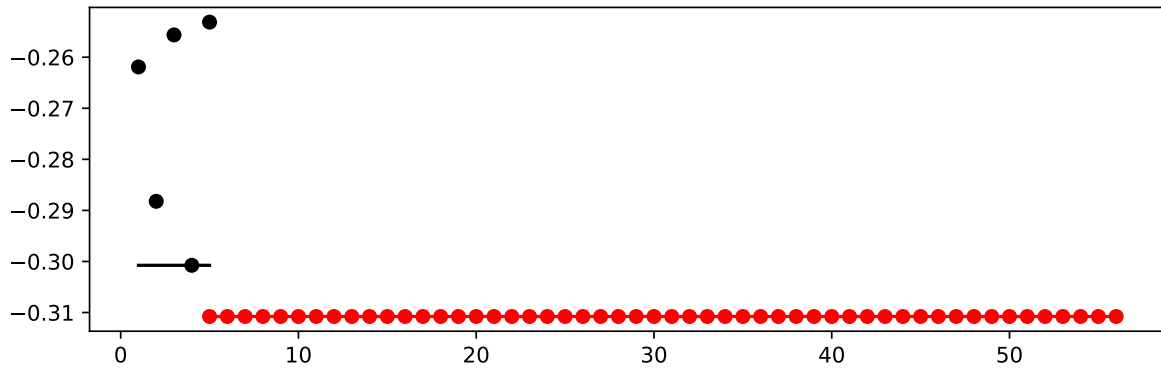



Figure 18.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- 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
```

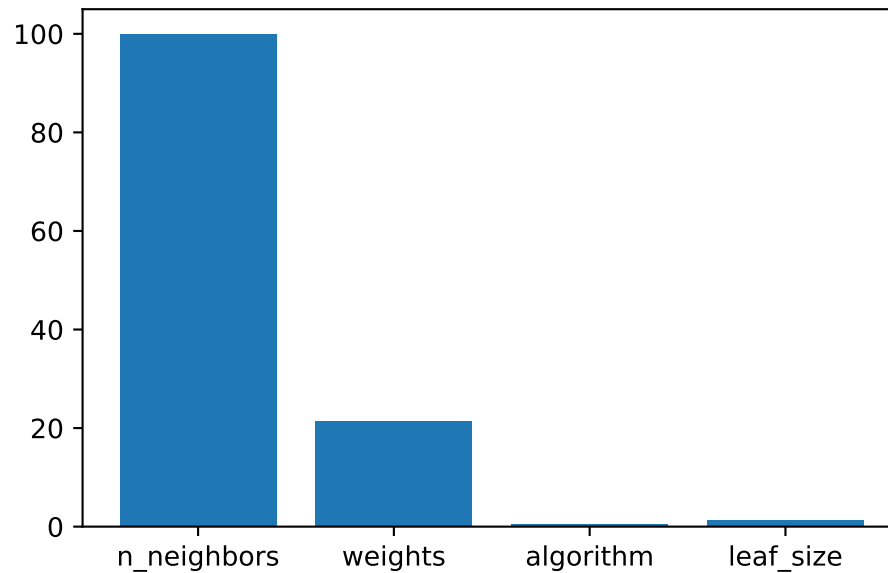


Figure 18.2: Variable importance plot, threshold 0.025.

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
```

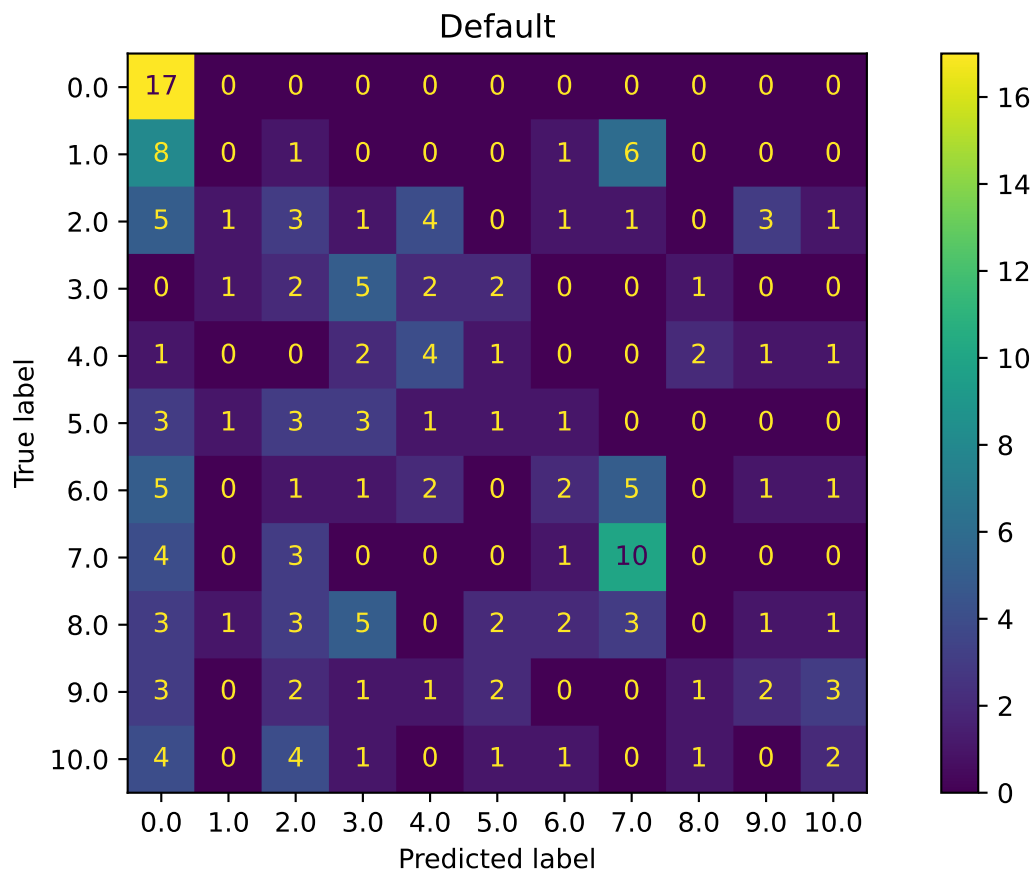
Since one single evaluation is not meaningful, we perform, similar to the evaluation of the SPOT results, $n = 30$ runs of the default setting and calculate the mean and standard deviation of the performance metric.

```
_ = repeated_eval(30, model_default)
```

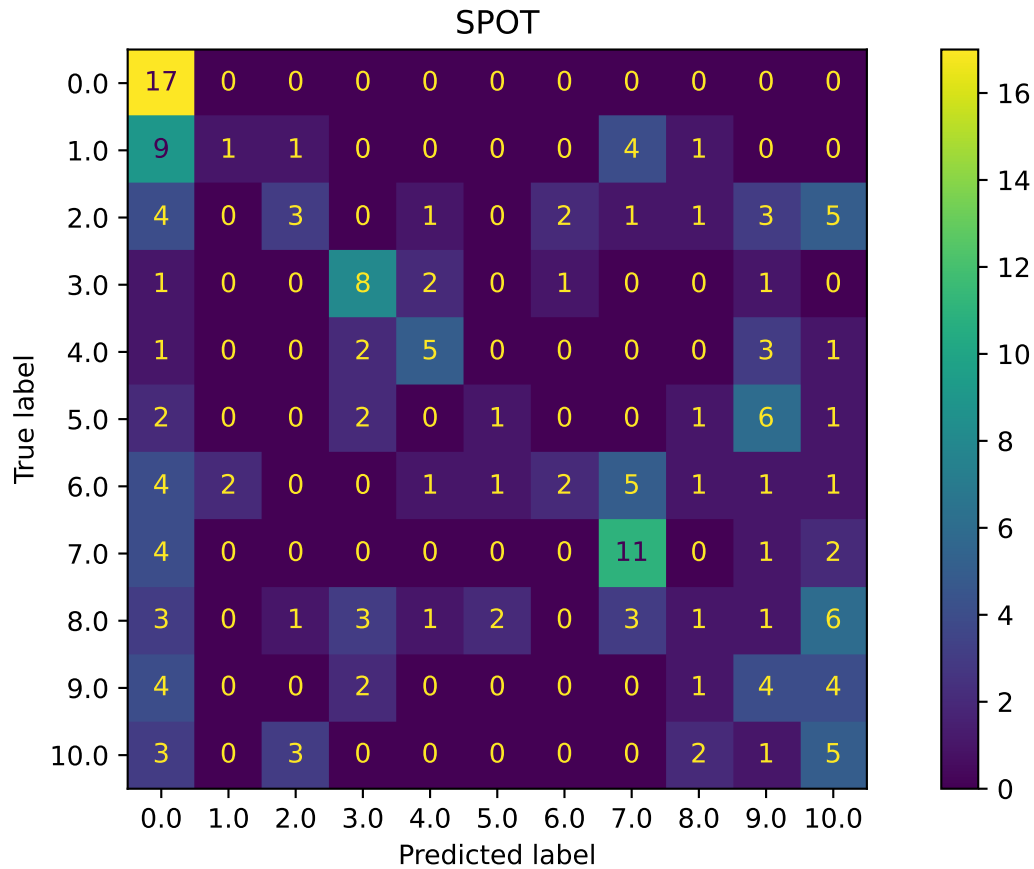
```
mean_res: 0.2768361581920903
std_res: 1.1102230246251565e-16
min_res: 0.2768361581920904
max_res: 0.2768361581920904
median_res: 0.2768361581920904
```

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

```

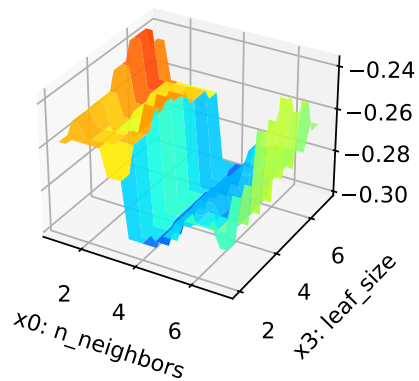
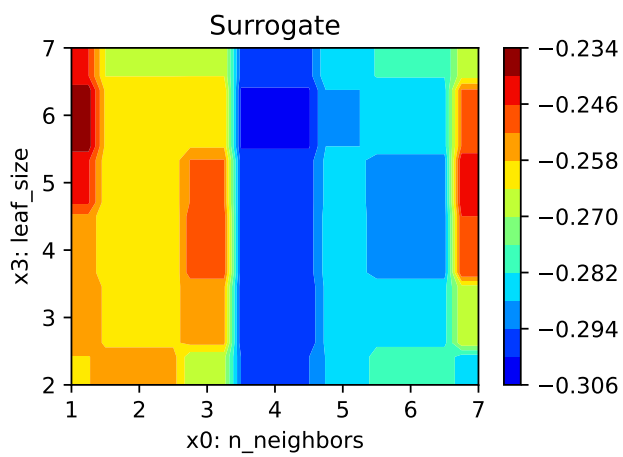
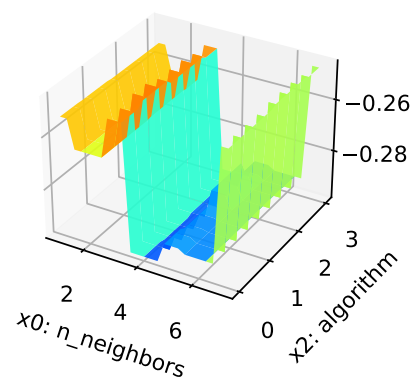
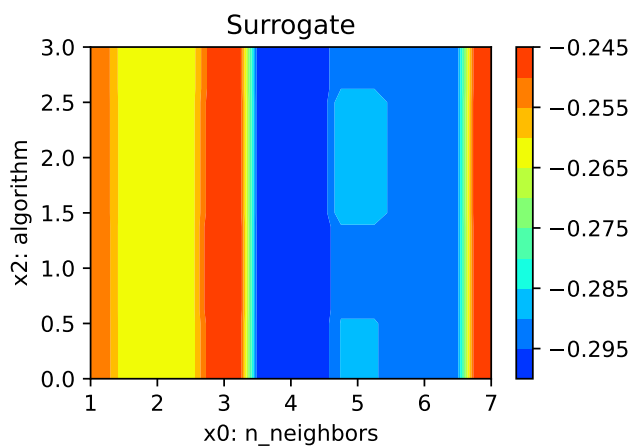
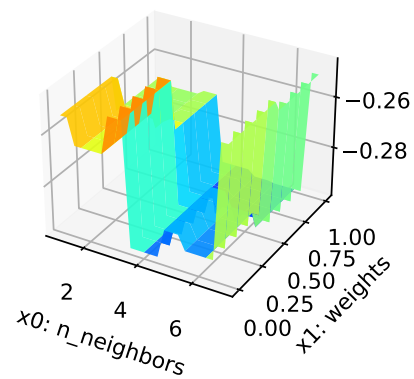
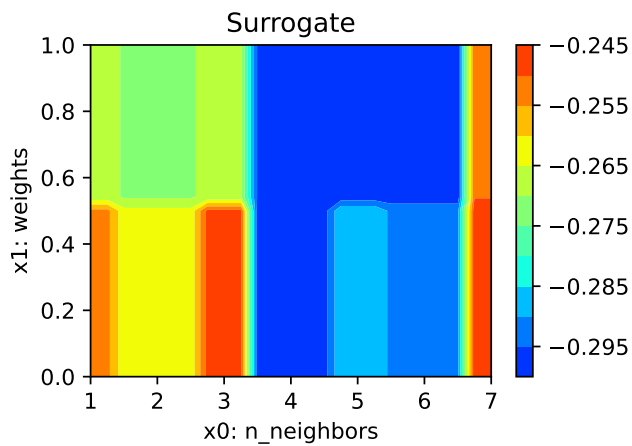
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)

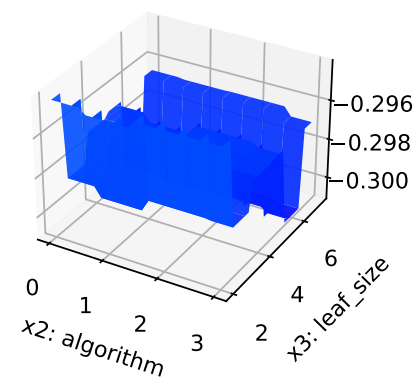
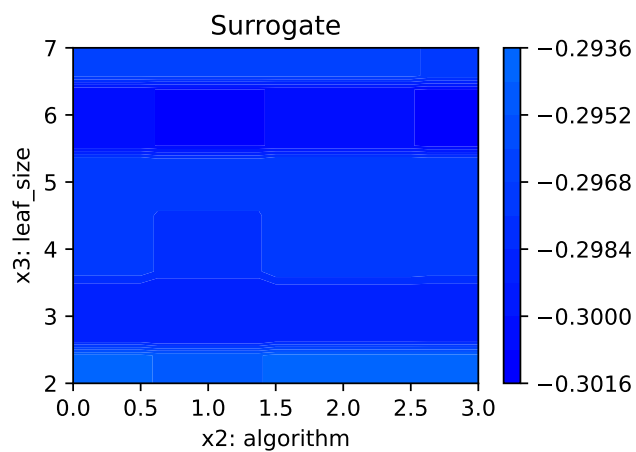
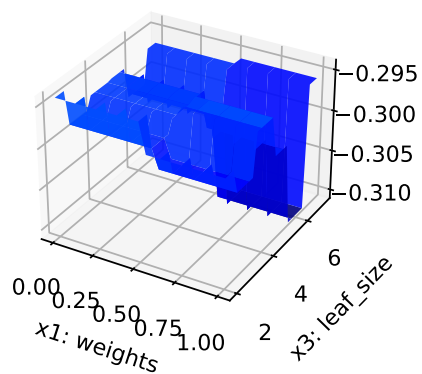
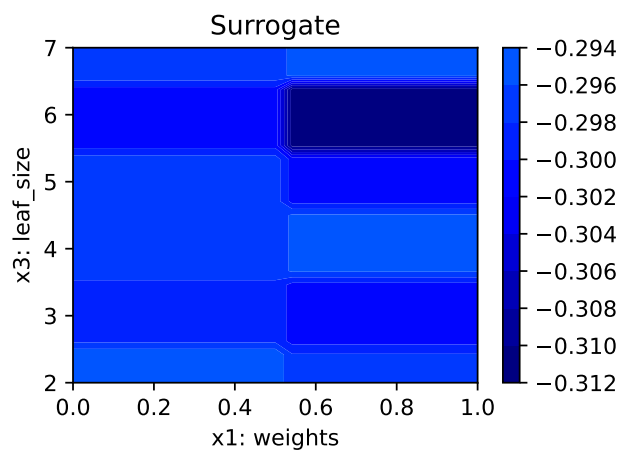
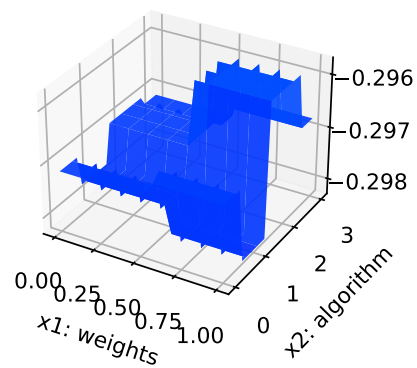
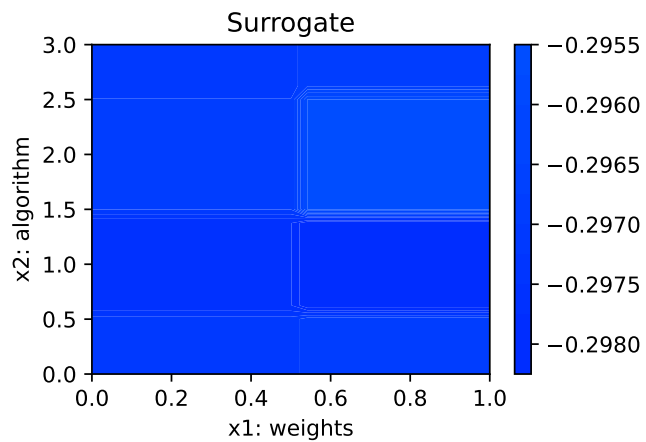
```

```

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	0.2.50
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_2023-06-28_04-36-25

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([[0.2368, 0.4273, 0.5062, 0.7959, 0.4734, 0.4461, 0.4880, 0.5386, 0.3882,
         0.6206],
        [0.6619, 0.2905, 0.5932, 0.4864, 0.2955, 0.5257, 0.5710, 0.6731, 0.6258,
         0.5196]])
tensor([0.4174, 0.4726])

```

- 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: 0.1763754934072495 | Loss: 0.0485901428925756 | Epoch: 2 | Mean
MeanAbsoluteError: 0.1494622379541397 | Loss: 0.0355353331879566 | Epoch: 4 | MeanAbsoluteEr
MeanAbsoluteError: 0.1548583507537842 | Loss: 0.0372416391282489 | Epoch: 8 | MeanAbsoluteEr
MeanAbsoluteError: 0.1184960529208183 | Loss: 0.0220915522659197 | Epoch: 12 | MeanAbsoluteEr

```

MeanAbsoluteError:	0.1117833182215691		Loss:	0.0196587825500357		Epoch:	16		MeanAbsoluteError:	0.1032603681087494
MeanAbsoluteError:	0.1032603681087494		Loss:	0.0173763246643112		Epoch:	20		MeanAbsoluteError:	0.0994351804256439
MeanAbsoluteError:	0.0994351804256439		Loss:	0.0151516693080530		Epoch:	24		MeanAbsoluteError:	0.0890197306871414
MeanAbsoluteError:	0.0890197306871414		Loss:	0.0142047490900088		Epoch:	28		MeanAbsoluteError:	0.0842181518673897
MeanAbsoluteError:	0.0842181518673897		Loss:	0.0117452974664047		Epoch:	32		MeanAbsoluteError:	0.0795542001724243
MeanAbsoluteError:	0.0795542001724243		Loss:	0.0115199759970174		Epoch:	36		MeanAbsoluteError:	0.0744432508945465
MeanAbsoluteError:	0.0744432508945465		Loss:	0.0099938525542568		Epoch:	40		MeanAbsoluteError:	0.0802907198667526
MeanAbsoluteError:	0.0802907198667526		Loss:	0.0110320067707155		Epoch:	44		MeanAbsoluteError:	0.0710762813687325
MeanAbsoluteError:	0.0710762813687325		Loss:	0.0090448455701239		Epoch:	48		MeanAbsoluteError:	0.0719680264592171
MeanAbsoluteError:	0.0719680264592171		Loss:	0.0090875115466157		Epoch:	52		MeanAbsoluteError:	0.0827575400471687
MeanAbsoluteError:	0.0827575400471687		Loss:	0.0103558856277040		Epoch:	56		MeanAbsoluteError:	0.0769331902265549
MeanAbsoluteError:	0.0769331902265549		Loss:	0.0103977151570450		Epoch:	60		MeanAbsoluteError:	0.0702800750732422
MeanAbsoluteError:	0.0702800750732422		Loss:	0.0094651353827335		Epoch:	64		MeanAbsoluteError:	0.0703848153352737
MeanAbsoluteError:	0.0703848153352737		Loss:	0.0084642412594373		Epoch:	68		MeanAbsoluteError:	0.0664932876825333
MeanAbsoluteError:	0.0664932876825333		Loss:	0.0075379565456196		Epoch:	72		MeanAbsoluteError:	0.0692544281482697
MeanAbsoluteError:	0.0692544281482697		Loss:	0.0094135811975177		Epoch:	76		MeanAbsoluteError:	0.0785929933190346
MeanAbsoluteError:	0.0785929933190346		Loss:	0.0103820906216769		Epoch:	80		MeanAbsoluteError:	0.0733307078480721
MeanAbsoluteError:	0.0733307078480721		Loss:	0.0104265510055580		Epoch:	84		MeanAbsoluteError:	

MeanAbsoluteError: 0.0669078826904297 | Loss: 0.0094460662160265 | Early stopping at epoch 8
Returned to Spot: Validation loss: 0.009446066216026483

config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.19981931523998656, 'lr_mult':
Epoch: 1 | MeanAbsoluteError: 0.3740435242652893 | Loss: 0.1640522864304091 | Epoch: 2 | Mean

MeanAbsoluteError: 0.1264226436614990 | Loss: 0.0238135681536637 | Epoch: 8 | MeanAbsoluteError:

MeanAbsoluteError: 0.1066688373684883 | Loss: 0.0172725177222961 | Epoch: 15 | MeanAbsoluteError:

MeanAbsoluteError: 0.0723838433623314 | Loss: 0.0079290293843338 | Epoch: 22 | MeanAbsoluteError:

MeanAbsoluteError: 0.0510691516101360 | Loss: 0.0047463981563372 | Epoch: 29 | MeanAbsoluteError:

MeanAbsoluteError: 0.0785520300269127 | Loss: 0.0085089306445106 | Epoch: 36 | MeanAbsoluteError:

MeanAbsoluteError: 0.0530484504997730 | Loss: 0.0051383535794326 | Epoch: 43 | MeanAbsoluteError:

MeanAbsoluteError: 0.0430076979100704 | Loss: 0.0034658647675410 | Epoch: 50 | MeanAbsoluteError:

MeanAbsoluteError: 0.0387333743274212 | Loss: 0.0028623487604292 | Epoch: 57 | MeanAbsoluteError:

MeanAbsoluteError: 0.0314101353287697 | Loss: 0.0023040296938760 | Epoch: 64 | MeanAbsoluteError:

MeanAbsoluteError: 0.0494104586541653 | Loss: 0.0041548571776060 | Epoch: 71 | MeanAbsoluteError:

MeanAbsoluteError: 0.0683930441737175 | Loss: 0.0080204120788135 | Epoch: 78 | MeanAbsoluteError:

MeanAbsoluteError: 0.0326718986034393 | Loss: 0.0023224149603936 | Epoch: 85 | MeanAbsoluteError:

MeanAbsoluteError: 0.0391912683844566 | Loss: 0.0027013285618619 | Epoch: 92 | MeanAbsoluteError:

MeanAbsoluteError: 0.0346203334629536 | Loss: 0.0022371264517699 | Epoch: 99 | MeanAbsoluteError:

MeanAbsoluteError: 0.0389983318746090 | Loss: 0.0028637222559681 | Epoch: 106 | MeanAbsoluteError:
Returned to Spot: Validation loss: 0.005246557172779974

config: {'_L_in': 10, '_L_out': 1, 'l1': 128, 'dropout_prob': 0.8582565260508446, 'lr_mult':
Epoch: 1 |

MeanAbsoluteError: 0.1906016021966934 | Loss: 0.0592176264765052 | Epoch: 2 |

MeanAbsoluteError: 0.1766470670700073 | Loss: 0.0488098074092219 | Epoch: 3 |

MeanAbsoluteError: 0.1557012200355530 | Loss: 0.0402734542052106 | Epoch: 4 |

MeanAbsoluteError: 0.1673262715339661 | Loss: 0.0419887646935725 | Epoch: 5 |

MeanAbsoluteError: 0.1458954960107803 | Loss: 0.0333697958201325 | Epoch: 6 |

MeanAbsoluteError: 0.1462909877300262 | Loss: 0.0352037123295789 | Epoch: 7 |

MeanAbsoluteError: 0.1493968367576599 | Loss: 0.0364414662531150 | Epoch: 8 |

MeanAbsoluteError: 0.1409311592578888 | Loss: 0.0316440720890144 | Epoch: 9 |

MeanAbsoluteError: 0.1387162506580353 | Loss: 0.0309447538327367 | Epoch: 10 |

MeanAbsoluteError: 0.1355140060186386 | Loss: 0.0292599441928905 | Epoch: 11 |

MeanAbsoluteError: 0.1329797953367233 | Loss: 0.0292514004903690 | Epoch: 12 |

MeanAbsoluteError: 0.1345205456018448 | Loss: 0.0296709148376249 | Epoch: 13 |

MeanAbsoluteError: 0.1360802650451660 | Loss: 0.0305579386909812 | Epoch: 14 |

MeanAbsoluteError: 0.1350934356451035 | Loss: 0.0295122445851060 | Epoch: 15 |

MeanAbsoluteError: 0.1360861063003540 | Loss: 0.0297027080138650 | Epoch: 16 |

MeanAbsoluteError: 0.1282459795475006 | Loss: 0.0273326403019989 | Epoch: 17 |

MeanAbsoluteError: 0.1337860673666000 | Loss: 0.0298419084923808 | Epoch: 18 |

MeanAbsoluteError: 0.1326735615730286 | Loss: 0.0293879085067116 | Epoch: 19 |

MeanAbsoluteError: 0.1399060189723969 | Loss: 0.0314165096310899 | Epoch: 20 |

MeanAbsoluteError: 0.1287903040647507 | Loss: 0.0279192590460783 | Epoch: 21 |

MeanAbsoluteError: 0.1306876689195633 | Loss: 0.0281696378471679 | Epoch: 22 |

MeanAbsoluteError: 0.1360104531049728 | Loss: 0.0301206801529922 | Epoch: 23 |

MeanAbsoluteError: 0.1371766924858093 | Loss: 0.0310429971360039 | Epoch: 24 |

MeanAbsoluteError: 0.1331977397203445 | Loss: 0.0287665841573228 | Epoch: 25 |

MeanAbsoluteError: 0.1345641762018204 | Loss: 0.0304510208707507 | Epoch: 26 |

MeanAbsoluteError: 0.1303422600030899 | Loss: 0.0289011441831826 | Epoch: 27 |

MeanAbsoluteError: 0.1289487779140472 | Loss: 0.0272149156402641 | Epoch: 28 |

MeanAbsoluteError: 0.1352220475673676 | Loss: 0.0292231386050116 | Epoch: 29 |

MeanAbsoluteError: 0.1306566148996353 | Loss: 0.0283692430909772 | Epoch: 30 |

MeanAbsoluteError: 0.1306436657905579 | Loss: 0.0281380402060070 | Epoch: 31 |

MeanAbsoluteError: 0.1316048353910446 | Loss: 0.0284060716220605 | Epoch: 32 |

MeanAbsoluteError: 0.1336109936237335 | Loss: 0.0292385677063915 | Epoch: 33 |

MeanAbsoluteError: 0.1342547237873077 | Loss: 0.0300507012194673 | Epoch: 34 |

MeanAbsoluteError: 0.1342622190713882 | Loss: 0.0292579571181856 | Epoch: 35 |

MeanAbsoluteError: 0.1313341408967972 | Loss: 0.0281139801340517 | Epoch: 36 |

MeanAbsoluteError: 0.1288254857063293 | Loss: 0.0281480011479653 | Epoch: 37 |

MeanAbsoluteError: 0.1308246552944183 | Loss: 0.0281029794123121 | Epoch: 38 |

MeanAbsoluteError: 0.1313728094100952 | Loss: 0.0293064752950643 | Epoch: 39 |

MeanAbsoluteError: 0.1298784762620926 | Loss: 0.0278203747002408 | Epoch: 40 |

MeanAbsoluteError: 0.1307501792907715 | Loss: 0.0282134012231836 | Epoch: 41 |

MeanAbsoluteError: 0.1298459172248840 | Loss: 0.0271049364501960 | Epoch: 42 |

MeanAbsoluteError: 0.1348453760147095 | Loss: 0.0286805420115707 | Epoch: 43 |

MeanAbsoluteError: 0.1334495991468430 | Loss: 0.0281898831684763 | Epoch: 44 |

MeanAbsoluteError: 0.1325608640909195 | Loss: 0.0280305280951628 | Epoch: 45 |

MeanAbsoluteError: 0.1296310573816299 | Loss: 0.0282381567567548 | Epoch: 46 |

MeanAbsoluteError: 0.1327182948589325 | Loss: 0.0288099643925671 | Epoch: 47 |

MeanAbsoluteError: 0.1361224055290222 | Loss: 0.0296891881222837 | Epoch: 48 |

MeanAbsoluteError: 0.1302625983953476 | Loss: 0.0277849604111786 | Epoch: 49 |

MeanAbsoluteError: 0.1268648356199265 | Loss: 0.0276235374849057 | Epoch: 50 |

MeanAbsoluteError: 0.1358816176652908 | Loss: 0.0295860600601494 | Epoch: 51 |

MeanAbsoluteError: 0.1339010745286942 | Loss: 0.0284944075753447 | Epoch: 52 |

MeanAbsoluteError: 0.1310700178146362 | Loss: 0.0283678813072523 | Epoch: 53 |

MeanAbsoluteError: 0.1327317804098129 | Loss: 0.0280746264711100 | Epoch: 54 |

MeanAbsoluteError: 0.1331279128789902 | Loss: 0.0291166906949547 | Epoch: 55 |

MeanAbsoluteError: 0.1326128989458084 | Loss: 0.0286119965320298 | Epoch: 56 |

MeanAbsoluteError: 0.1301606893539429 | Loss: 0.0279647060402203 | Epoch: 57 |

MeanAbsoluteError: 0.1299051940441132 | Loss: 0.0279882040596931 | Epoch: 58 |

MeanAbsoluteError: 0.1320620924234390 | Loss: 0.0286330333579099 | Epoch: 59 |

MeanAbsoluteError: 0.1328964680433273 | Loss: 0.0281104689711841 | Epoch: 60 |

MeanAbsoluteError: 0.1312973648309708 | Loss: 0.0278395807067864 | Epoch: 61 |

MeanAbsoluteError: 0.1304374337196350 | Loss: 0.0275492594700578 | Epoch: 62 |

MeanAbsoluteError: 0.1277497261762619 | Loss: 0.0277401072687159 | Epoch: 63 |

MeanAbsoluteError: 0.1279569715261459 | Loss: 0.0268374334751085 | Epoch: 64 |

MeanAbsoluteError: 0.1337484270334244 | Loss: 0.0282759450397377 | Epoch: 65 |

MeanAbsoluteError: 0.1279855817556381 | Loss: 0.0268031686648707 | Epoch: 66 |

MeanAbsoluteError: 0.1311047077178955 | Loss: 0.0284825343186943 | Epoch: 67 |

MeanAbsoluteError: 0.1290601789951324 | Loss: 0.0280340483061930 | Epoch: 68 |

MeanAbsoluteError: 0.1294728517532349 | Loss: 0.0267499092816918 | Epoch: 69 |

MeanAbsoluteError: 0.1309904158115387 | Loss: 0.0285915042957398 | Epoch: 70 |

MeanAbsoluteError: 0.1325741857290268 | Loss: 0.0291585051691315 | Epoch: 71 |

MeanAbsoluteError: 0.1271493136882782 | Loss: 0.0269686307563264 | Epoch: 72 |

MeanAbsoluteError: 0.1322593986988068 | Loss: 0.0287028482961371 | Epoch: 73 |

MeanAbsoluteError: 0.1282913535833359 | Loss: 0.0277975844642303 | Epoch: 74 |

MeanAbsoluteError: 0.1309762001037598 | Loss: 0.0277644140702372 | Epoch: 75 |

MeanAbsoluteError: 0.1307172030210495 | Loss: 0.0272801836077269 | Epoch: 76 |

MeanAbsoluteError: 0.1294242888689041 | Loss: 0.0271462069436287 | Epoch: 77 |

MeanAbsoluteError: 0.1296325773000717 | Loss: 0.0277659982497183 | Epoch: 78 |

MeanAbsoluteError: 0.1261638253927231 | Loss: 0.0274722182568803 | Epoch: 79 |

MeanAbsoluteError: 0.1276392340660095 | Loss: 0.0275245589178424 | Epoch: 80 |

MeanAbsoluteError: 0.1263939589262009 | Loss: 0.0267094437710087 | Epoch: 81 |

MeanAbsoluteError: 0.1315495371818542 | Loss: 0.0281920114341968 | Epoch: 82 |

MeanAbsoluteError: 0.1280868351459503 | Loss: 0.0266295598738361 | Epoch: 83 |

MeanAbsoluteError: 0.1256980299949646 | Loss: 0.0265254787027758 | Epoch: 84 |

MeanAbsoluteError: 0.1312648504972458 | Loss: 0.0271897905970885 | Epoch: 85 |

MeanAbsoluteError: 0.1255203932523727 | Loss: 0.0267312342583500 | Epoch: 86 |

MeanAbsoluteError: 0.1264547258615494 | Loss: 0.0268778088205727 | Epoch: 87 |

MeanAbsoluteError: 0.1254378706216812 | Loss: 0.0273740415429832 | Epoch: 88 |

MeanAbsoluteError: 0.1285453587770462 | Loss: 0.0270927341975524 | Epoch: 89 |

MeanAbsoluteError: 0.1242725253105164 | Loss: 0.0260941678449550 | Epoch: 90 |

MeanAbsoluteError: 0.1330325901508331 | Loss: 0.0294300170545087 | Epoch: 91 |

MeanAbsoluteError: 0.1282961666584015 | Loss: 0.0278217022314372 | Epoch: 92 |

MeanAbsoluteError: 0.1296312958002090 | Loss: 0.0270231985795544 | Epoch: 93 |

MeanAbsoluteError: 0.1270213276147842 | Loss: 0.0265486166121264 | Epoch: 94 |

MeanAbsoluteError: 0.1292451024055481 | Loss: 0.0270955564506585 | Epoch: 95 |

MeanAbsoluteError: 0.1297568082809448 | Loss: 0.0262789687580759 | Epoch: 96 |

MeanAbsoluteError: 0.1297270506620407 | Loss: 0.0270707196672811 | Epoch: 97 |

MeanAbsoluteError: 0.1294361799955368 | Loss: 0.0274995277525159 | Epoch: 98 |

MeanAbsoluteError: 0.1291230320930481 | Loss: 0.0274793036372527 | Epoch: 99 |

MeanAbsoluteError: 0.1302778869867325 | Loss: 0.0275899459198384 | Epoch: 100 |

MeanAbsoluteError: 0.1310527175664902 | Loss: 0.0271826518870754 | Epoch: 101 |

MeanAbsoluteError: 0.1284277588129044 | Loss: 0.0274415362164533 | Epoch: 102 |

MeanAbsoluteError: 0.1250814348459244 | Loss: 0.0264028706246366 | Epoch: 103 |

MeanAbsoluteError: 0.1300929486751556 | Loss: 0.0276186662560334 | Epoch: 104 |

MeanAbsoluteError: 0.1291172206401825 | Loss: 0.0266493249518680 | Epoch: 105 |

MeanAbsoluteError: 0.1289844959974289 | Loss: 0.0268343811109147 | Epoch: 106 |

MeanAbsoluteError: 0.1282078027725220 | Loss: 0.0269936819809906 | Epoch: 107 |

MeanAbsoluteError: 0.1312871426343918 | Loss: 0.0268861653254135 | Epoch: 108 |

MeanAbsoluteError: 0.1279205828905106 | Loss: 0.0270266057959331 | Epoch: 109 |

MeanAbsoluteError: 0.1265664994716644 | Loss: 0.0253741085151948 | Epoch: 110 |

MeanAbsoluteError: 0.1240428164601326 | Loss: 0.0260692550689176 | Epoch: 111 |

MeanAbsoluteError: 0.1291914135217667 | Loss: 0.0274001824513834 | Epoch: 112 |

MeanAbsoluteError: 0.1265890449285507 | Loss: 0.0256999918524040 | Epoch: 113 |

MeanAbsoluteError: 0.1289791166782379 | Loss: 0.0270870484857975 | Epoch: 114 |

MeanAbsoluteError: 0.1290157139301300 | Loss: 0.0275732101115864 | Epoch: 115 |

MeanAbsoluteError: 0.1316838264465332 | Loss: 0.0285116187048455 | Epoch: 116 |

MeanAbsoluteError: 0.1243665218353271 | Loss: 0.0254087239445653 | Epoch: 117 |

MeanAbsoluteError: 0.1301486641168594 | Loss: 0.0276620389505600 | Epoch: 118 |

MeanAbsoluteError: 0.1248610913753510 | Loss: 0.0265419108570101 | Epoch: 119 |

MeanAbsoluteError: 0.1295434385538101 | Loss: 0.0271625900169602 | Epoch: 120 |

MeanAbsoluteError: 0.1263412386178970 | Loss: 0.0263201591568456 | Epoch: 121 |

MeanAbsoluteError: 0.1278229355812073 | Loss: 0.0266937810070037 | Epoch: 122 |

MeanAbsoluteError: 0.1214506626129150 | Loss: 0.0252790291180524 | Epoch: 123 |

MeanAbsoluteError: 0.1203084066510201 | Loss: 0.0248256546027066 | Epoch: 124 |

MeanAbsoluteError: 0.1226054355502129 | Loss: 0.0244701935930304 | Epoch: 125 |

MeanAbsoluteError: 0.1287318915128708 | Loss: 0.0276610208296916 | Epoch: 126 |

MeanAbsoluteError: 0.1289600431919098 | Loss: 0.0265105011320823 | Epoch: 127 |

MeanAbsoluteError: 0.1268023997545242 | Loss: 0.0262618637667038 | Epoch: 128 |

MeanAbsoluteError: 0.1304685622453690 | Loss: 0.0278044908134810 | Epoch: 129 |

MeanAbsoluteError: 0.1286353468894958 | Loss: 0.0275906383525580 | Epoch: 130 |

MeanAbsoluteError: 0.1247602105140686 | Loss: 0.0256985403333480 | Epoch: 131 |

MeanAbsoluteError: 0.1253454685211182 | Loss: 0.0262414728593528 | Epoch: 132 |

MeanAbsoluteError: 0.1270460039377213 | Loss: 0.0253817854474376 | Epoch: 133 |

MeanAbsoluteError: 0.1271986961364746 | Loss: 0.0260650650532625 | Epoch: 134 |

MeanAbsoluteError: 0.1298877149820328 | Loss: 0.0267961457879816 | Epoch: 135 |

MeanAbsoluteError: 0.1276598572731018 | Loss: 0.0276636828557821 | Epoch: 136 |

MeanAbsoluteError: 0.1265471875667572 | Loss: 0.0255945388380981 | Epoch: 137 |

MeanAbsoluteError: 0.1264865547418594 | Loss: 0.0257028157454139 | Epoch: 138 |

MeanAbsoluteError: 0.1252900809049606 | Loss: 0.0263132125472790 | Epoch: 139 |

MeanAbsoluteError: 0.1293774992227554 | Loss: 0.0265454614826983 | Epoch: 140 |

MeanAbsoluteError: 0.1234815195202827 | Loss: 0.0251943679576895 | Epoch: 141 |

MeanAbsoluteError: 0.1204348653554916 | Loss: 0.0234962729826414 | Epoch: 142 |

MeanAbsoluteError: 0.1213308572769165 | Loss: 0.0242404112817288 | Epoch: 143 |

MeanAbsoluteError: 0.1249863803386688 | Loss: 0.0251679350115592 | Epoch: 144 |

MeanAbsoluteError: 0.1205008476972580 | Loss: 0.0238228082475446 | Epoch: 145 |

MeanAbsoluteError: 0.1240228265523911 | Loss: 0.0252927108372872 | Epoch: 146 |

MeanAbsoluteError: 0.1258285194635391 | Loss: 0.0255119406719071 | Epoch: 147 |

MeanAbsoluteError: 0.1255249679088593 | Loss: 0.0254526315901118 | Epoch: 148 |

MeanAbsoluteError: 0.1290921717882156 | Loss: 0.0267898270271447 | Epoch: 149 |

MeanAbsoluteError: 0.1201840862631798 | Loss: 0.0238122622616356 | Epoch: 150 |

MeanAbsoluteError: 0.1237462088465691 | Loss: 0.0251034539557683 | Epoch: 151 |

MeanAbsoluteError: 0.1276439130306244 | Loss: 0.0258439982581573 | Epoch: 152 |

MeanAbsoluteError: 0.1252454966306686 | Loss: 0.0256014420028562 | Epoch: 153 |

MeanAbsoluteError: 0.1283676922321320 | Loss: 0.0269177570206618 | Epoch: 154 |

MeanAbsoluteError: 0.1236772015690804 | Loss: 0.0248442271687478 | Epoch: 155 |

MeanAbsoluteError: 0.1270201802253723 | Loss: 0.0269170428256621 | Epoch: 156 |

MeanAbsoluteError: 0.1230964660644531 | Loss: 0.0247944148716245 | Epoch: 157 |

MeanAbsoluteError: 0.1279508918523788 | Loss: 0.0263091822580706 | Epoch: 158 |

MeanAbsoluteError: 0.1238981857895851 | Loss: 0.0254940125148278 | Epoch: 159 |

MeanAbsoluteError: 0.1233379244804382 | Loss: 0.0252547345248361 | Epoch: 160 |

MeanAbsoluteError: 0.1294707804918289 | Loss: 0.0272145976137836 | Epoch: 161 |

MeanAbsoluteError: 0.1249440535902977 | Loss: 0.0251262179395417 | Epoch: 162 |

MeanAbsoluteError: 0.1237394586205482 | Loss: 0.0251606964096815 | Epoch: 163 |

MeanAbsoluteError: 0.1238756179809570 | Loss: 0.0253581405206205 | Epoch: 164 |

MeanAbsoluteError: 0.1228734552860260 | Loss: 0.0251325946118838 | Epoch: 165 |

MeanAbsoluteError: 0.1260121166706085 | Loss: 0.0258746383648152 | Epoch: 166 |

MeanAbsoluteError: 0.1219793930649757 | Loss: 0.0238197015847618 | Epoch: 167 |

MeanAbsoluteError: 0.1263855993747711 | Loss: 0.0260729016574624 | Epoch: 168 |

MeanAbsoluteError: 0.1259359717369080 | Loss: 0.0252560991470818 | Epoch: 169 |

MeanAbsoluteError: 0.1281655430793762 | Loss: 0.0272475174402158 | Epoch: 170 |

MeanAbsoluteError: 0.1203510761260986 | Loss: 0.0235094136385305 | Epoch: 171 |

MeanAbsoluteError: 0.1245295330882072 | Loss: 0.0254221304718885 | Epoch: 172 |

MeanAbsoluteError: 0.1197978481650352 | Loss: 0.0235344512020432 | Epoch: 173 |

MeanAbsoluteError: 0.1262005418539047 | Loss: 0.0255987150499520 | Epoch: 174 |

MeanAbsoluteError: 0.1297573149204254 | Loss: 0.0267036704239824 | Epoch: 175 |

MeanAbsoluteError: 0.1240839660167694 | Loss: 0.0246928096936123 | Epoch: 176 |

MeanAbsoluteError: 0.1254555284976959 | Loss: 0.0264209409902105 | Epoch: 177 |

MeanAbsoluteError: 0.1268683224916458 | Loss: 0.0258531176879963 | Epoch: 178 |

MeanAbsoluteError: 0.1235556006431580 | Loss: 0.0256750437341786 | Epoch: 179 |

MeanAbsoluteError: 0.1248909756541252 | Loss: 0.0249832368681867 | Epoch: 180 |

MeanAbsoluteError: 0.1239279210567474 | Loss: 0.0244012493585857 | Epoch: 181 |

MeanAbsoluteError: 0.1265070438385010 | Loss: 0.0260955071352722 | Epoch: 182 |

MeanAbsoluteError: 0.1212122216820717 | Loss: 0.0237844942672806 | Epoch: 183 |

MeanAbsoluteError: 0.1247469335794449 | Loss: 0.0245881366040946 | Epoch: 184 |

MeanAbsoluteError: 0.1267666518688202 | Loss: 0.0271111738336428 | Epoch: 185 |

MeanAbsoluteError: 0.1243599355220795 | Loss: 0.0252539903538369 | Epoch: 186 |

MeanAbsoluteError: 0.1258383393287659 | Loss: 0.0252702450046005 | Epoch: 187 |

MeanAbsoluteError: 0.1248923987150192 | Loss: 0.0246179323852994 | Epoch: 188 |

MeanAbsoluteError: 0.1226476952433586 | Loss: 0.0244668135576891 | Epoch: 189 |

MeanAbsoluteError: 0.1238370761275291 | Loss: 0.0244073042628588 | Epoch: 190 |

MeanAbsoluteError: 0.1289105415344238 | Loss: 0.0271023318066970 | Epoch: 191 |

MeanAbsoluteError: 0.1189232394099236 | Loss: 0.0241569651607036 | Epoch: 192 |

MeanAbsoluteError: 0.1220492571592331 | Loss: 0.0238054507899020 | Epoch: 193 |

MeanAbsoluteError: 0.1200881600379944 | Loss: 0.0228718761291263 | Epoch: 194 |

MeanAbsoluteError: 0.1235086694359779 | Loss: 0.0246276608721625 | Epoch: 195 |

MeanAbsoluteError: 0.1199718713760376 | Loss: 0.0237334734894102 | Epoch: 196 |

MeanAbsoluteError: 0.1210648566484451 | Loss: 0.0243828867455159 | Epoch: 197 |

MeanAbsoluteError: 0.1235227361321449 | Loss: 0.0246497023490216 | Epoch: 198 |

MeanAbsoluteError: 0.1203418225049973 | Loss: 0.0240384035820413 | Epoch: 199 |

MeanAbsoluteError: 0.1242450103163719 | Loss: 0.0258431979629677 | Epoch: 200 |

MeanAbsoluteError: 0.1252431571483612 | Loss: 0.0252355232763997 | Epoch: 201 |

MeanAbsoluteError: 0.1176738217473030 | Loss: 0.0232555406351457 | Epoch: 202 |

MeanAbsoluteError: 0.1237546652555466 | Loss: 0.0255173344180124 | Epoch: 203 |

MeanAbsoluteError: 0.1226902678608894 | Loss: 0.0245023775378165 | Epoch: 204 |

MeanAbsoluteError: 0.1254051923751831 | Loss: 0.0260029403804704 | Epoch: 205 |

MeanAbsoluteError: 0.1246930658817291 | Loss: 0.0247389293426871 | Epoch: 206 |

MeanAbsoluteError: 0.1201661825180054 | Loss: 0.0238803099773941 | Epoch: 207 |

MeanAbsoluteError: 0.1239907592535019 | Loss: 0.0240203872992424 | Epoch: 208 |

MeanAbsoluteError: 0.1226023361086845 | Loss: 0.0248857348399664 | Epoch: 209 |

MeanAbsoluteError: 0.1203230172395706 | Loss: 0.0244307663668587 | Epoch: 210 |

MeanAbsoluteError: 0.1220517084002495 | Loss: 0.0238008851752481 | Epoch: 211 |

MeanAbsoluteError: 0.1235695108771324 | Loss: 0.0248091122050149 | Epoch: 212 |

MeanAbsoluteError: 0.1164538338780403 | Loss: 0.0231802657026371 | Epoch: 213 |

MeanAbsoluteError: 0.1201070398092270 | Loss: 0.0235392870094317 | Epoch: 214 |

MeanAbsoluteError: 0.1204570755362511 | Loss: 0.0225916114130329 | Epoch: 215 |

MeanAbsoluteError: 0.1217629089951515 | Loss: 0.0240348582187774 | Epoch: 216 |

MeanAbsoluteError: 0.1266159415245056 | Loss: 0.0260992669570260 | Epoch: 217 |

MeanAbsoluteError: 0.1202245578169823 | Loss: 0.0232311633809392 | Epoch: 218 |

MeanAbsoluteError: 0.1228091195225716 | Loss: 0.0240701731722462 | Epoch: 219 |

MeanAbsoluteError: 0.1200103089213371 | Loss: 0.0233730163824900 | Epoch: 220 |

MeanAbsoluteError: 0.1236013397574425 | Loss: 0.0237725362604639 | Epoch: 221 |

MeanAbsoluteError: 0.1199375167489052 | Loss: 0.0227292551694942 | Epoch: 222 |

MeanAbsoluteError: 0.1227657198905945 | Loss: 0.0242221099214051 | Epoch: 223 |

MeanAbsoluteError: 0.1164323166012764 | Loss: 0.0221535399898615 | Epoch: 224 |

MeanAbsoluteError: 0.1191984042525291 | Loss: 0.0234889744655963 | Epoch: 225 |

MeanAbsoluteError: 0.1225371658802032 | Loss: 0.0235242710224702 | Epoch: 226 |

MeanAbsoluteError: 0.1302139461040497 | Loss: 0.0266055087166994 | Epoch: 227 |

MeanAbsoluteError: 0.1207338497042656 | Loss: 0.0238872882628736 | Epoch: 228 |

MeanAbsoluteError: 0.1158916950225830 | Loss: 0.0218304336876220 | Epoch: 229 |

MeanAbsoluteError: 0.1236096173524857 | Loss: 0.0242348659709872 | Epoch: 230 |

MeanAbsoluteError: 0.1259942501783371 | Loss: 0.0255064621783094 | Epoch: 231 |

MeanAbsoluteError: 0.1206161975860596 | Loss: 0.0232471501360124 | Epoch: 232 |

MeanAbsoluteError: 0.1156412735581398 | Loss: 0.0215316253128428 | Epoch: 233 |

MeanAbsoluteError: 0.1242966428399086 | Loss: 0.0254553905500810 | Epoch: 234 |

MeanAbsoluteError: 0.1190652996301651 | Loss: 0.0228935582902826 | Epoch: 235 |

MeanAbsoluteError: 0.1221973672509193 | Loss: 0.0235509585733719 | Epoch: 236 |

MeanAbsoluteError: 0.1199862435460091 | Loss: 0.0244859041060166 | Epoch: 237 |

MeanAbsoluteError: 0.1202658489346504 | Loss: 0.0238071400535409 | Epoch: 238 |

MeanAbsoluteError: 0.1203504204750061 | Loss: 0.0236616945789622 | Epoch: 239 |

MeanAbsoluteError: 0.1257338225841522 | Loss: 0.0258282554255978 | Epoch: 240 |

MeanAbsoluteError: 0.1251066178083420 | Loss: 0.0251666437218470 | Epoch: 241 |

MeanAbsoluteError: 0.1235644817352295 | Loss: 0.0246201962582321 | Epoch: 242 |

MeanAbsoluteError: 0.1221055090427399 | Loss: 0.0242509750490232 | Epoch: 243 |

MeanAbsoluteError: 0.1190812662243843 | Loss: 0.0227265113236839 | Epoch: 244 |

MeanAbsoluteError: 0.1134565323591232 | Loss: 0.0208000766637269 | Epoch: 245 |

MeanAbsoluteError: 0.1183220818638802 | Loss: 0.0229925046913801 | Epoch: 246 |

MeanAbsoluteError: 0.1279331147670746 | Loss: 0.0261302097393491 | Epoch: 247 |

MeanAbsoluteError: 0.1139709502458572 | Loss: 0.0209668377868365 | Epoch: 248 |

MeanAbsoluteError: 0.1190543100237846 | Loss: 0.0231220262308610 | Epoch: 249 |

MeanAbsoluteError: 0.1201552227139473 | Loss: 0.0240623459829173 | Epoch: 250 |

MeanAbsoluteError: 0.1189419403672218 | Loss: 0.0242874761075169 | Epoch: 251 |

MeanAbsoluteError: 0.1127973943948746 | Loss: 0.0217591625404627 | Epoch: 252 |

MeanAbsoluteError: 0.1145754903554916 | Loss: 0.0223257315957744 | Epoch: 253 |

MeanAbsoluteError: 0.1212803125381470 | Loss: 0.0235781962985675 | Epoch: 254 |

MeanAbsoluteError: 0.1200234740972519 | Loss: 0.0235292481768799 | Epoch: 255 |

MeanAbsoluteError: 0.1173524335026741 | Loss: 0.0228756317310642 | Epoch: 256 |

MeanAbsoluteError: 0.1166131570935249 | Loss: 0.0226863101963439 | Epoch: 257 |

MeanAbsoluteError: 0.1186512112617493 | Loss: 0.0240411521729887 | Epoch: 258 |

MeanAbsoluteError: 0.1182444617152214 | Loss: 0.0235351999934452 | Epoch: 259 |

MeanAbsoluteError: 0.1170909404754639 | Loss: 0.0238537219642694 | Epoch: 260 |

MeanAbsoluteError: 0.1213305145502090 | Loss: 0.0231882864908160 | Epoch: 261 |

MeanAbsoluteError: 0.1130168884992599 | Loss: 0.0209253678156529 | Epoch: 262 |

MeanAbsoluteError: 0.1167898178100586 | Loss: 0.0226991572701081 | Epoch: 263 |

MeanAbsoluteError: 0.1192471086978912 | Loss: 0.0225000221871596 | Epoch: 264 |

MeanAbsoluteError: 0.1194991916418076 | Loss: 0.0233197889682682 | Epoch: 265 |

MeanAbsoluteError: 0.1219474375247955 | Loss: 0.0236199068541464 | Epoch: 266 |

MeanAbsoluteError: 0.1181627437472343 | Loss: 0.0229242804500488 | Epoch: 267 |

MeanAbsoluteError: 0.1112342700362206 | Loss: 0.0198621940676821 | Epoch: 268 |

MeanAbsoluteError: 0.1208035498857498 | Loss: 0.0238604194919268 | Epoch: 269 |

MeanAbsoluteError: 0.1209155544638634 | Loss: 0.0241818949941080 | Epoch: 270 |

MeanAbsoluteError: 0.1201035305857658 | Loss: 0.0230163025438863 | Epoch: 271 |

MeanAbsoluteError: 0.1183257177472115 | Loss: 0.0222561745834537 | Epoch: 272 |

MeanAbsoluteError: 0.1157606020569801 | Loss: 0.0227046667075289 | Epoch: 273 |

MeanAbsoluteError: 0.1183840781450272 | Loss: 0.0223672621988226 | Epoch: 274 |

MeanAbsoluteError: 0.1196970269083977 | Loss: 0.0239160013033082 | Epoch: 275 |

MeanAbsoluteError: 0.1161367148160934 | Loss: 0.0224632055778056 | Epoch: 276 |

MeanAbsoluteError: 0.1203264594078064 | Loss: 0.0238980766930278 | Epoch: 277 |

MeanAbsoluteError: 0.1212601065635681 | Loss: 0.0239861658323207 | Epoch: 278 |

MeanAbsoluteError: 0.1190326437354088 | Loss: 0.0234844347836527 | Epoch: 279 |

MeanAbsoluteError: 0.1160259991884232 | Loss: 0.0210213431518544 | Epoch: 280 |

MeanAbsoluteError: 0.1223167702555656 | Loss: 0.0232966716740581 | Epoch: 281 |

MeanAbsoluteError: 0.1183132156729698 | Loss: 0.0236418176277463 | Epoch: 282 |

MeanAbsoluteError: 0.1136554852128029 | Loss: 0.0216984093138793 | Epoch: 283 |

MeanAbsoluteError: 0.1169446781277657 | Loss: 0.0222611998089512 | Epoch: 284 |

MeanAbsoluteError: 0.1096244156360626 | Loss: 0.0206578080606657 | Epoch: 285 |

MeanAbsoluteError: 0.1191040575504303 | Loss: 0.0232607657064606 | Epoch: 286 |

MeanAbsoluteError: 0.1140528097748756 | Loss: 0.0207875116166057 | Epoch: 287 |

MeanAbsoluteError: 0.1127698421478271 | Loss: 0.0213103612452202 | Epoch: 288 |

MeanAbsoluteError: 0.1171211525797844 | Loss: 0.0227392583427718 | Epoch: 289 |

MeanAbsoluteError: 0.1180902197957039 | Loss: 0.0227314350624025 | Epoch: 290 |

MeanAbsoluteError: 0.1135296523571014 | Loss: 0.0208337499931592 | Epoch: 291 |

MeanAbsoluteError: 0.1145100370049477 | Loss: 0.0226575872375057 | Epoch: 292 |

MeanAbsoluteError: 0.1207895427942276 | Loss: 0.0241423986188602 | Epoch: 293 |

MeanAbsoluteError: 0.1135959997773170 | Loss: 0.0214877617592962 | Epoch: 294 |

MeanAbsoluteError: 0.1209083572030067 | Loss: 0.0239827092249955 | Epoch: 295 |

MeanAbsoluteError: 0.1175181046128273 | Loss: 0.0218775299606690 | Epoch: 296 |

MeanAbsoluteError: 0.1156006604433060 | Loss: 0.0216130465044989 | Epoch: 297 |

MeanAbsoluteError: 0.1160327419638634 | Loss: 0.0219900224031395 | Epoch: 298 |

MeanAbsoluteError: 0.1113632842898369 | Loss: 0.0205715814839641 | Epoch: 299 |

MeanAbsoluteError: 0.1180929020047188 | Loss: 0.0226509888938259 | Epoch: 300 |

MeanAbsoluteError: 0.1059642285108566 | Loss: 0.0181246412278657 | Epoch: 301 |

MeanAbsoluteError: 0.1174105927348137 | Loss: 0.0231362721818065 | Epoch: 302 |

MeanAbsoluteError: 0.1151966974139214 | Loss: 0.0215217774793079 | Epoch: 303 |

MeanAbsoluteError: 0.1107075139880180 | Loss: 0.0195050651603439 | Epoch: 304 |

MeanAbsoluteError: 0.1142964959144592 | Loss: 0.0214332855456329 | Epoch: 305 |

MeanAbsoluteError: 0.1161567196249962 | Loss: 0.0218778810413399 | Epoch: 306 |

MeanAbsoluteError: 0.1176167279481888 | Loss: 0.0224442206456357 | Epoch: 307 |

MeanAbsoluteError: 0.1151985898613930 | Loss: 0.0220375953349139 | Epoch: 308 |

MeanAbsoluteError: 0.1149947121739388 | Loss: 0.0211353497372086 | Epoch: 309 |

MeanAbsoluteError: 0.1146838665008545 | Loss: 0.0209840899892151 | Epoch: 310 |

MeanAbsoluteError: 0.1067757308483124 | Loss: 0.0191716891845378 | Epoch: 311 |

MeanAbsoluteError: 0.1123808026313782 | Loss: 0.0206060062837787 | Epoch: 312 |

MeanAbsoluteError: 0.1103892028331757 | Loss: 0.0199456037981630 | Epoch: 313 |

MeanAbsoluteError: 0.1077160239219666 | Loss: 0.0197689023279236 | Epoch: 314 |

MeanAbsoluteError: 0.1123124286532402 | Loss: 0.0214754926741201 | Epoch: 315 |

MeanAbsoluteError: 0.1184544488787651 | Loss: 0.0232608367349045 | Epoch: 316 |

MeanAbsoluteError: 0.1142841950058937 | Loss: 0.0215352745215932 | Epoch: 317 |

MeanAbsoluteError: 0.1187190786004066 | Loss: 0.0218270168364446 | Epoch: 318 |

MeanAbsoluteError: 0.1162487640976906 | Loss: 0.0218634434511963 | Epoch: 319 |

MeanAbsoluteError: 0.1163908019661903 | Loss: 0.0219870057843703 | Epoch: 320 |

MeanAbsoluteError: 0.1171950548887253 | Loss: 0.0224385351930929 | Epoch: 321 |

MeanAbsoluteError: 0.1101758480072021 | Loss: 0.0201999060868790 | Epoch: 322 |

MeanAbsoluteError: 0.1115751489996910 | Loss: 0.0210910531593254 | Epoch: 323 |

MeanAbsoluteError: 0.1124361529946327 | Loss: 0.0205635421840755 | Epoch: 324 |

MeanAbsoluteError: 0.1175003200769424 | Loss: 0.0223858018185820 | Epoch: 325 |

MeanAbsoluteError: 0.1138163357973099 | Loss: 0.0205475822675119 | Epoch: 326 |

MeanAbsoluteError: 0.1124817132949829 | Loss: 0.0208085702829946 | Epoch: 327 |

MeanAbsoluteError: 0.1068047955632210 | Loss: 0.0197435060716998 | Epoch: 328 |

MeanAbsoluteError: 0.1094501391053200 | Loss: 0.0198744315048680 | Epoch: 329 |

MeanAbsoluteError: 0.1169408187270164 | Loss: 0.0217184968356257 | Epoch: 330 |

MeanAbsoluteError: 0.1141101866960526 | Loss: 0.0213178470363103 | Epoch: 331 |

MeanAbsoluteError: 0.1148832440376282 | Loss: 0.0213130130895782 | Epoch: 332 |

MeanAbsoluteError: 0.1146366596221924 | Loss: 0.0218351742551264 | Epoch: 333 |

MeanAbsoluteError: 0.1117613762617111 | Loss: 0.0206630486768942 | Epoch: 334 |

MeanAbsoluteError: 0.1113478466868401 | Loss: 0.0202932688018579 | Epoch: 335 |

MeanAbsoluteError: 0.1083755120635033 | Loss: 0.0191945153900209 | Epoch: 336 |

MeanAbsoluteError: 0.1084419861435890 | Loss: 0.0191655761344100 | Epoch: 337 |

MeanAbsoluteError: 0.1158087402582169 | Loss: 0.0210433183331043 | Epoch: 338 |

MeanAbsoluteError: 0.1109479516744614 | Loss: 0.0204269988629191 | Epoch: 339 |

MeanAbsoluteError: 0.1057877987623215 | Loss: 0.0195785306761718 | Epoch: 340 |

MeanAbsoluteError: 0.1113590076565742 | Loss: 0.0212080688937810 | Epoch: 341 |

MeanAbsoluteError: 0.1121371835470200 | Loss: 0.0198364759081839 | Epoch: 342 |

MeanAbsoluteError: 0.1120048686861992 | Loss: 0.0207966410466179 | Epoch: 343 |

MeanAbsoluteError: 0.1154108047485352 | Loss: 0.0216558271071820 | Epoch: 344 |

MeanAbsoluteError: 0.1107658669352531 | Loss: 0.0206396006097445 | Epoch: 345 |

MeanAbsoluteError: 0.1122847497463226 | Loss: 0.0204901131859515 | Epoch: 346 |

MeanAbsoluteError: 0.1152684912085533 | Loss: 0.0219429557009911 | Epoch: 347 |

MeanAbsoluteError: 0.1141020804643631 | Loss: 0.0213041185586189 | Epoch: 348 |

MeanAbsoluteError: 0.1117976009845734 | Loss: 0.0207107552875580 | Epoch: 349 |

MeanAbsoluteError: 0.1121278777718544 | Loss: 0.0206236782125537 | Epoch: 350 |

MeanAbsoluteError: 0.1142685860395432 | Loss: 0.0215147685919267 | Epoch: 351 |

MeanAbsoluteError: 0.1152022406458855 | Loss: 0.0215723813628805 | Epoch: 352 |

MeanAbsoluteError: 0.1140899509191513 | Loss: 0.0215571154219409 | Epoch: 353 |

MeanAbsoluteError: 0.1155313625931740 | Loss: 0.0214319556282135 | Epoch: 354 |

MeanAbsoluteError: 0.1142012774944305 | Loss: 0.0218317611156090 | Epoch: 355 |

MeanAbsoluteError: 0.1156949251890182 | Loss: 0.0218745871146772 | Epoch: 356 |

MeanAbsoluteError: 0.1142719909548759 | Loss: 0.0220089183192855 | Epoch: 357 |

MeanAbsoluteError: 0.1097313538193703 | Loss: 0.0201771644838542 | Epoch: 358 |

MeanAbsoluteError: 0.1105767190456390 | Loss: 0.0203035693157532 | Epoch: 359 |

MeanAbsoluteError: 0.1175991967320442 | Loss: 0.0221507545129134 | Epoch: 360 |

MeanAbsoluteError: 0.1145573705434799 | Loss: 0.0207079107136815 | Epoch: 361 |

MeanAbsoluteError: 0.1049721837043762 | Loss: 0.0185833706260413 | Epoch: 362 |

MeanAbsoluteError: 0.1147904321551323 | Loss: 0.0213407516238901 | Epoch: 363 |

MeanAbsoluteError: 0.1167801767587662 | Loss: 0.0221992349618328 | Epoch: 364 |

MeanAbsoluteError: 0.1137582138180733 | Loss: 0.0214401967604257 | Epoch: 365 |

MeanAbsoluteError: 0.1064267680048943 | Loss: 0.0184261988162810 | Epoch: 366 |

MeanAbsoluteError: 0.1096132546663284 | Loss: 0.0200683041645001 | Epoch: 367 |

MeanAbsoluteError: 0.1105989217758179 | Loss: 0.0206624933257505 | Epoch: 368 |

MeanAbsoluteError: 0.1123722344636917 | Loss: 0.0197771896553362 | Epoch: 369 |

MeanAbsoluteError: 0.1160574629902840 | Loss: 0.0214978471018064 | Epoch: 370 |

MeanAbsoluteError: 0.1106129065155983 | Loss: 0.0200183531820464 | Epoch: 371 |

MeanAbsoluteError: 0.1165467053651810 | Loss: 0.0221224252443668 | Epoch: 372 |

MeanAbsoluteError: 0.1131957098841667 | Loss: 0.0198883614575607 | Epoch: 373 |

MeanAbsoluteError: 0.1187211871147156 | Loss: 0.0226113250787360 | Epoch: 374 |

MeanAbsoluteError: 0.1110255196690559 | Loss: 0.0199963685846114 | Epoch: 375 |

MeanAbsoluteError: 0.1147969290614128 | Loss: 0.0217144255073314 | Epoch: 376 |

MeanAbsoluteError: 0.1106549575924873 | Loss: 0.0198606012253246 | Epoch: 377 |

MeanAbsoluteError: 0.1028734743595123 | Loss: 0.0175832485106609 | Epoch: 378 |

MeanAbsoluteError: 0.1172391995787621 | Loss: 0.0235830223121836 | Epoch: 379 |

MeanAbsoluteError: 0.1102108508348465 | Loss: 0.0198883109405021 | Epoch: 380 |

MeanAbsoluteError: 0.1049054116010666 | Loss: 0.0187539176485249 | Epoch: 381 |

MeanAbsoluteError: 0.1156074032187462 | Loss: 0.0218566492447765 | Epoch: 382 |

MeanAbsoluteError: 0.1048213988542557 | Loss: 0.0191066752872090 | Epoch: 383 |

MeanAbsoluteError: 0.1098940297961235 | Loss: 0.0208059251748152 | Epoch: 384 |

MeanAbsoluteError: 0.1047836020588875 | Loss: 0.0192152717353504 | Epoch: 385 |

MeanAbsoluteError: 0.1097176223993301 | Loss: 0.0201249030288939 | Epoch: 386 |

MeanAbsoluteError: 0.1020346581935883 | Loss: 0.0172220743935516 | Epoch: 387 |

MeanAbsoluteError: 0.1091000512242317 | Loss: 0.0209513811220434 | Epoch: 388 |

MeanAbsoluteError: 0.1096540838479996 | Loss: 0.0199946520436303 | Epoch: 389 |

MeanAbsoluteError: 0.1055795401334763 | Loss: 0.0185369547520046 | Epoch: 390 |

MeanAbsoluteError: 0.1170183569192886 | Loss: 0.0219742145084213 | Epoch: 391 |

MeanAbsoluteError: 0.1034271046519279 | Loss: 0.0170891361248262 | Epoch: 392 |

MeanAbsoluteError: 0.1149630621075630 | Loss: 0.0210577200386130 | Epoch: 393 |

MeanAbsoluteError: 0.1147883757948875 | Loss: 0.0205910712628975 | Epoch: 394 |

MeanAbsoluteError: 0.1152223348617554 | Loss: 0.0206024630032092 | Epoch: 395 |

MeanAbsoluteError: 0.1065103933215141 | Loss: 0.0193731863397018 | Epoch: 396 |

MeanAbsoluteError: 0.1076340898871422 | Loss: 0.0193604778354832 | Epoch: 397 |

MeanAbsoluteError: 0.1085976511240005 | Loss: 0.0198051471026095 | Epoch: 398 |

MeanAbsoluteError: 0.1089780032634735 | Loss: 0.0197342881228542 | Epoch: 399 |

MeanAbsoluteError: 0.1114269793033600 | Loss: 0.0210821832794075 | Epoch: 400 |

MeanAbsoluteError: 0.1033895835280418 | Loss: 0.0179107633315531 | Epoch: 401 |

MeanAbsoluteError: 0.1092498376965523 | Loss: 0.0192945396745927 | Epoch: 402 |

MeanAbsoluteError: 0.1132700070738792 | Loss: 0.0200787355472532 | Epoch: 403 |

MeanAbsoluteError: 0.1099159717559814 | Loss: 0.0195796035367918 | Epoch: 404 |

MeanAbsoluteError: 0.1116530224680901 | Loss: 0.0196163749543484 | Epoch: 405 |

MeanAbsoluteError: 0.1042973548173904 | Loss: 0.0181809429000835 | Epoch: 406 |

MeanAbsoluteError: 0.1137545555830002 | Loss: 0.0209585832181619 | Epoch: 407 |

MeanAbsoluteError: 0.1065363287925720 | Loss: 0.0189945588752623 | Epoch: 408 |

MeanAbsoluteError: 0.1082545369863510 | Loss: 0.0193379129760433 | Epoch: 409 |

MeanAbsoluteError: 0.1036268696188927 | Loss: 0.0184587751810614 | Epoch: 410 |

MeanAbsoluteError: 0.1125273033976555 | Loss: 0.0203532895438063 | Epoch: 411 |

MeanAbsoluteError: 0.1029620319604874 | Loss: 0.0182009271124844 | Epoch: 412 |

MeanAbsoluteError: 0.1146038845181465 | Loss: 0.0206200136335489 | Epoch: 413 |

MeanAbsoluteError: 0.1158820688724518 | Loss: 0.0221018793548865 | Epoch: 414 |

MeanAbsoluteError: 0.1129317209124565 | Loss: 0.0208273101997232 | Epoch: 415 |

MeanAbsoluteError: 0.1070851758122444 | Loss: 0.0195039466590121 | Epoch: 416 |

MeanAbsoluteError: 0.1106814295053482 | Loss: 0.0209799519511095 | Epoch: 417 |

MeanAbsoluteError: 0.1076225638389587 | Loss: 0.0195644125342369 | Epoch: 418 |

MeanAbsoluteError: 0.1128773987293243 | Loss: 0.0203147566963647 | Epoch: 419 |

MeanAbsoluteError: 0.1106478273868561 | Loss: 0.0200710849585206 | Epoch: 420 |

MeanAbsoluteError: 0.1042694002389908 | Loss: 0.0181450751535400 | Epoch: 421 |

MeanAbsoluteError: 0.1083867251873016 | Loss: 0.0197194014992177 | Epoch: 422 |

MeanAbsoluteError: 0.1134244799613953 | Loss: 0.0204319706551178 | Epoch: 423 |

MeanAbsoluteError: 0.1079911068081856 | Loss: 0.0182074064895278 | Epoch: 424 |

MeanAbsoluteError: 0.1098677068948746 | Loss: 0.0201051457110467 | Epoch: 425 |

MeanAbsoluteError: 0.1118015125393867 | Loss: 0.0206291160362889 | Epoch: 426 |

MeanAbsoluteError: 0.1176157221198082 | Loss: 0.0219779512868263 | Epoch: 427 |

MeanAbsoluteError: 0.1089806631207466 | Loss: 0.0187065692181932 | Epoch: 428 |

MeanAbsoluteError: 0.1119714900851250 | Loss: 0.0204901625762189 | Epoch: 429 |

MeanAbsoluteError: 0.1026374623179436 | Loss: 0.0173813790623353 | Epoch: 430 |

MeanAbsoluteError: 0.1028376743197441 | Loss: 0.0173975813437816 | Epoch: 431 |

MeanAbsoluteError: 0.1069479882717133 | Loss: 0.0189526435843436 | Epoch: 432 |

MeanAbsoluteError: 0.1080096587538719 | Loss: 0.0189928838595612 | Epoch: 433 |

MeanAbsoluteError: 0.1001797765493393 | Loss: 0.0169016758605145 | Epoch: 434 |

MeanAbsoluteError: 0.1036572232842445 | Loss: 0.0179398333892338 | Epoch: 435 |

MeanAbsoluteError: 0.1057966947555542 | Loss: 0.0177641858898035 | Epoch: 436 |

MeanAbsoluteError: 0.1033790558576584 | Loss: 0.0173190728670549 | Epoch: 437 |

MeanAbsoluteError: 0.1070581972599030 | Loss: 0.0189675544570916 | Epoch: 438 |

MeanAbsoluteError: 0.1102360785007477 | Loss: 0.0195077043983717 | Epoch: 439 |

MeanAbsoluteError: 0.1078652963042259 | Loss: 0.0182829698889206 | Epoch: 440 |

MeanAbsoluteError: 0.1077211797237396 | Loss: 0.0190865745873210 | Epoch: 441 |

MeanAbsoluteError: 0.1105160266160965 | Loss: 0.0192597272344089 | Epoch: 442 |

MeanAbsoluteError: 0.1085066497325897 | Loss: 0.0194539795065066 | Epoch: 443 |

MeanAbsoluteError: 0.1063431277871132 | Loss: 0.0188389568576531 | Epoch: 444 |

MeanAbsoluteError: 0.1059265807271004 | Loss: 0.0181368090101751 | Epoch: 445 |

MeanAbsoluteError: 0.1119832322001457 | Loss: 0.0197560402254506 | Epoch: 446 |

MeanAbsoluteError: 0.1008085086941719 | Loss: 0.0177223905962698 | Epoch: 447 |

MeanAbsoluteError: 0.1079691350460052 | Loss: 0.0197127060119722 | Epoch: 448 |

MeanAbsoluteError: 0.1147835701704025 | Loss: 0.0212169444446287 | Epoch: 449 |

MeanAbsoluteError: 0.1065463870763779 | Loss: 0.0190592707174073 | Epoch: 450 |

MeanAbsoluteError: 0.1019655764102936 | Loss: 0.0172277661472132 | Epoch: 451 |

MeanAbsoluteError: 0.0997239500284195 | Loss: 0.0169527390626414 | Epoch: 452 |

MeanAbsoluteError: 0.1073995083570480 | Loss: 0.0190608450163078 | Epoch: 453 |

MeanAbsoluteError: 0.1162973046302795 | Loss: 0.0206393253547139 | Epoch: 454 |

MeanAbsoluteError: 0.1022050529718399 | Loss: 0.0177555750142104 | Epoch: 455 |

MeanAbsoluteError: 0.1082983911037445 | Loss: 0.0185306362768946 | Epoch: 456 |

MeanAbsoluteError: 0.1056950166821480 | Loss: 0.0181809969501289 | Epoch: 457 |

MeanAbsoluteError: 0.1033701673150063 | Loss: 0.0185771323887942 | Epoch: 458 |

MeanAbsoluteError: 0.1034902110695839 | Loss: 0.0170608163474390 | Epoch: 459 |

MeanAbsoluteError: 0.1021550819277763 | Loss: 0.0172560390316842 | Epoch: 460 |

MeanAbsoluteError: 0.1041354164481163 | Loss: 0.0181417646435875 | Epoch: 461 |

MeanAbsoluteError: 0.1068131402134895 | Loss: 0.0183424594825677 | Epoch: 462 |

MeanAbsoluteError: 0.1048176735639572 | Loss: 0.0187176132976310 | Epoch: 463 |

MeanAbsoluteError: 0.1008830592036247 | Loss: 0.0180215356963163 | Epoch: 464 |

MeanAbsoluteError: 0.1034813076257706 | Loss: 0.0174423857861742 | Epoch: 465 |

MeanAbsoluteError: 0.0998788550496101 | Loss: 0.0167609283582230 | Epoch: 466 |

MeanAbsoluteError: 0.1072045639157295 | Loss: 0.0193971683951774 | Epoch: 467 |

MeanAbsoluteError: 0.1036697924137115 | Loss: 0.0178104932358383 | Epoch: 468 |

MeanAbsoluteError: 0.1087950542569160 | Loss: 0.0189240616663786 | Epoch: 469 |

MeanAbsoluteError: 0.1040836125612259 | Loss: 0.0180448388054477 | Epoch: 470 |

MeanAbsoluteError: 0.1028655469417572 | Loss: 0.0181250653663786 | Epoch: 471 |

MeanAbsoluteError: 0.1064008101820946 | Loss: 0.0180494776476553 | Epoch: 472 |

MeanAbsoluteError: 0.1096077859401703 | Loss: 0.0194758526366180 | Epoch: 473 |

MeanAbsoluteError: 0.1039750352501869 | Loss: 0.0178360439534299 | Epoch: 474 |

MeanAbsoluteError: 0.1147305667400360 | Loss: 0.0206917664447489 | Epoch: 475 |

MeanAbsoluteError: 0.1006496772170067 | Loss: 0.0171782939586168 | Epoch: 476 |

MeanAbsoluteError: 0.1036782264709473 | Loss: 0.0180763221115437 | Epoch: 477 |

MeanAbsoluteError: 0.1073877438902855 | Loss: 0.0189647845766740 | Epoch: 478 |

MeanAbsoluteError: 0.1033189967274666 | Loss: 0.0179820420132213 | Epoch: 479 |

MeanAbsoluteError: 0.0969226807355881 | Loss: 0.0151110131417469 | Epoch: 480 |

MeanAbsoluteError: 0.1011137887835503 | Loss: 0.0167527064099219 | Epoch: 481 |

MeanAbsoluteError: 0.1109265908598900 | Loss: 0.0197683482486173 | Epoch: 482 |

MeanAbsoluteError: 0.0943513140082359 | Loss: 0.0143740005955139 | Epoch: 483 |

MeanAbsoluteError: 0.1052556037902832 | Loss: 0.0182136284494367 | Epoch: 484 |

MeanAbsoluteError: 0.1049083545804024 | Loss: 0.0172649666072296 | Epoch: 485 |

MeanAbsoluteError: 0.1000338494777679 | Loss: 0.0174022773884644 | Epoch: 486 |

MeanAbsoluteError: 0.1041606217622757 | Loss: 0.0186230261239689 | Epoch: 487 |

MeanAbsoluteError: 0.1056728810071945 | Loss: 0.0183020952756488 | Epoch: 488 |

MeanAbsoluteError: 0.1052759587764740 | Loss: 0.0184052071753831 | Epoch: 489 |

MeanAbsoluteError: 0.1073449552059174 | Loss: 0.0188967828929890 | Epoch: 490 |

MeanAbsoluteError: 0.1051058620214462 | Loss: 0.0183355943631614 | Epoch: 491 |

MeanAbsoluteError: 0.0992389693856239 | Loss: 0.0163649649316600 | Epoch: 492 |

MeanAbsoluteError: 0.1006151959300041 | Loss: 0.0168978297910993 | Epoch: 493 |

MeanAbsoluteError: 0.1007028371095657 | Loss: 0.0164990701826173 | Epoch: 494 |

MeanAbsoluteError: 0.1000699326395988 | Loss: 0.0166905564716308 | Epoch: 495 |

MeanAbsoluteError: 0.1031717732548714 | Loss: 0.0172749300506545 | Epoch: 496 |

MeanAbsoluteError: 0.1071192324161530 | Loss: 0.0176077892470494 | Epoch: 497 |

MeanAbsoluteError: 0.1070759072899818 | Loss: 0.0177249297114516 | Epoch: 498 |

MeanAbsoluteError: 0.1005775779485703 | Loss: 0.0168534656256209 | Epoch: 499 |

MeanAbsoluteError: 0.1036857590079308 | Loss: 0.0170621620826205 | Epoch: 500 |

MeanAbsoluteError: 0.1010491922497749 | Loss: 0.0167988000290400 | Epoch: 501 |

MeanAbsoluteError: 0.1050731465220451 | Loss: 0.0187990766849058 | Epoch: 502 |

MeanAbsoluteError: 0.1048061922192574 | Loss: 0.0180612353284543 | Epoch: 503 |

MeanAbsoluteError: 0.1041964367032051 | Loss: 0.0181424163291619 | Epoch: 504 |

MeanAbsoluteError: 0.1069685742259026 | Loss: 0.0180598690484718 | Epoch: 505 |

MeanAbsoluteError: 0.1060271114110947 | Loss: 0.0180286342552669 | Epoch: 506 |

MeanAbsoluteError: 0.1073452979326248 | Loss: 0.0189377885528666 | Epoch: 507 |

MeanAbsoluteError: 0.0991455912590027 | Loss: 0.0159714572002607 | Epoch: 508 |

MeanAbsoluteError: 0.1015143096446991 | Loss: 0.0169697173347231 | Epoch: 509 |

MeanAbsoluteError: 0.1034625768661499 | Loss: 0.0178569539097953 | Epoch: 510 |

MeanAbsoluteError: 0.1029700562357903 | Loss: 0.0170899799219721 | Epoch: 511 |

MeanAbsoluteError: 0.1064960286021233 | Loss: 0.0185391758134756 | Epoch: 512 |

MeanAbsoluteError: 0.1048054024577141 | Loss: 0.0180370159226004 | Epoch: 513 |

MeanAbsoluteError: 0.1056984141469002 | Loss: 0.0181526078504006 | Epoch: 514 |

MeanAbsoluteError: 0.1013862714171410 | Loss: 0.0177615212877087 | Epoch: 515 |

MeanAbsoluteError: 0.0954622775316238 | Loss: 0.0155976643086494 | Epoch: 516 |

MeanAbsoluteError: 0.0987756252288818 | Loss: 0.0160607120638512 | Epoch: 517 |

MeanAbsoluteError: 0.1002181097865105 | Loss: 0.0161289276681297 | Epoch: 518 |

MeanAbsoluteError: 0.1015105322003365 | Loss: 0.0184959332480018 | Epoch: 519 |

MeanAbsoluteError: 0.1041478589177132 | Loss: 0.0176783003479068 | Epoch: 520 |

MeanAbsoluteError: 0.0964286848902702 | Loss: 0.0155873401807185 | Epoch: 521 |

MeanAbsoluteError: 0.1003126427531242 | Loss: 0.0164738670745055 | Epoch: 522 |

MeanAbsoluteError: 0.1053038910031319 | Loss: 0.0187589548227212 | Epoch: 523 |

MeanAbsoluteError: 0.1018480360507965 | Loss: 0.0178072257226449 | Epoch: 524 |

MeanAbsoluteError: 0.1026825532317162 | Loss: 0.0169282552146251 | Epoch: 525 |

MeanAbsoluteError: 0.1114047989249229 | Loss: 0.0200619970195112 | Epoch: 526 |

MeanAbsoluteError: 0.0931172892451286 | Loss: 0.0151107468726029 | Epoch: 527 |

MeanAbsoluteError: 0.1045416742563248 | Loss: 0.0177867583516248 | Epoch: 528 |

MeanAbsoluteError: 0.1016073077917099 | Loss: 0.0166905939717738 | Epoch: 529 |

MeanAbsoluteError: 0.1056850552558899 | Loss: 0.0184278271289077 | Epoch: 530 |

MeanAbsoluteError: 0.1021522730588913 | Loss: 0.0176627519953763 | Epoch: 531 |

MeanAbsoluteError: 0.1070622131228447 | Loss: 0.0186486646975391 | Epoch: 532 |

MeanAbsoluteError: 0.1003908514976501 | Loss: 0.0161873221281348 | Epoch: 533 |

MeanAbsoluteError: 0.1003976315259933 | Loss: 0.0167665838136842 | Epoch: 534 |

MeanAbsoluteError: 0.0984870791435242 | Loss: 0.0170537861441941 | Epoch: 535 |

MeanAbsoluteError: 0.0997051522135735 | Loss: 0.0162957796826170 | Epoch: 536 |

MeanAbsoluteError: 0.1033106669783592 | Loss: 0.0172143290161815 | Epoch: 537 |

MeanAbsoluteError: 0.1013860777020454 | Loss: 0.0174771010941671 | Epoch: 538 |

MeanAbsoluteError: 0.1038577035069466 | Loss: 0.0169330651940739 | Epoch: 539 |

MeanAbsoluteError: 0.0990650728344917 | Loss: 0.0163618415728949 | Epoch: 540 |

MeanAbsoluteError: 0.0956544950604439 | Loss: 0.0153050115861697 | Epoch: 541 |

MeanAbsoluteError: 0.1007461994886398 | Loss: 0.0165602440692601 | Epoch: 542 |

MeanAbsoluteError: 0.1037358567118645 | Loss: 0.0174919494097412 | Epoch: 543 |

MeanAbsoluteError: 0.0981289520859718 | Loss: 0.0164385300782305 | Epoch: 544 |

MeanAbsoluteError: 0.1021715849637985 | Loss: 0.0172414022553251 | Epoch: 545 |

MeanAbsoluteError: 0.0973268002271652 | Loss: 0.0154736636909850 | Epoch: 546 |

MeanAbsoluteError: 0.1081335321068764 | Loss: 0.0194958307992783 | Epoch: 547 |

MeanAbsoluteError: 0.0973451957106590 | Loss: 0.0155605739896661 | Epoch: 548 |

MeanAbsoluteError: 0.0994981527328491 | Loss: 0.0178956933877635 | Epoch: 549 |

MeanAbsoluteError: 0.1022071093320847 | Loss: 0.0173080166181656 | Epoch: 550 |

MeanAbsoluteError: 0.0971766188740730 | Loss: 0.0151706127515839 | Epoch: 551 |

MeanAbsoluteError: 0.0987697765231133 | Loss: 0.0160272676935877 | Epoch: 552 |

MeanAbsoluteError: 0.0970997363328934 | Loss: 0.0157509209737570 | Epoch: 553 |

MeanAbsoluteError: 0.1044185012578964 | Loss: 0.0178223420546419 | Epoch: 554 |

MeanAbsoluteError: 0.1030491441488266 | Loss: 0.0179536996787647 | Epoch: 555 |

MeanAbsoluteError: 0.0984480008482933 | Loss: 0.0154872239356822 | Epoch: 556 |

MeanAbsoluteError: 0.0963018611073494 | Loss: 0.0152608611386192 | Epoch: 557 |

MeanAbsoluteError: 0.0994376689195633 | Loss: 0.0163923761249104 | Epoch: 558 |

MeanAbsoluteError: 0.0945814251899719 | Loss: 0.0149690753822021 | Epoch: 559 |

MeanAbsoluteError: 0.1023883372545242 | Loss: 0.0171672590870276 | Epoch: 560 |

MeanAbsoluteError: 0.0965294763445854 | Loss: 0.0157735080182708 | Epoch: 561 |

MeanAbsoluteError: 0.0986420512199402 | Loss: 0.0171576174530977 | Epoch: 562 |

MeanAbsoluteError: 0.1015981584787369 | Loss: 0.0167020819132449 | Epoch: 563 |

MeanAbsoluteError: 0.1013983264565468 | Loss: 0.0170930780556713 | Epoch: 564 |

MeanAbsoluteError: 0.0983861237764359 | Loss: 0.0161811664618047 | Epoch: 565 |

MeanAbsoluteError: 0.0954068303108215 | Loss: 0.0154653397136841 | Epoch: 566 |

MeanAbsoluteError: 0.0932306274771690 | Loss: 0.0138596284792099 | Epoch: 567 |

MeanAbsoluteError: 0.1004961654543877 | Loss: 0.0171452726035932 | Epoch: 568 |

MeanAbsoluteError: 0.0965415909886360 | Loss: 0.0159225437008233 | Epoch: 569 |

MeanAbsoluteError: 0.0997540652751923 | Loss: 0.0164659726698786 | Epoch: 570 |

MeanAbsoluteError: 0.1032027006149292 | Loss: 0.0175669107126305 | Epoch: 571 |

MeanAbsoluteError: 0.0974706038832664 | Loss: 0.0161494104043231 | Epoch: 572 |

MeanAbsoluteError: 0.1041113063693047 | Loss: 0.0190396069939986 | Epoch: 573 |

MeanAbsoluteError: 0.1024399176239967 | Loss: 0.0168527208725573 | Epoch: 574 |

MeanAbsoluteError: 0.0988530591130257 | Loss: 0.0157731307465777 | Epoch: 575 |

MeanAbsoluteError: 0.1036386266350746 | Loss: 0.0180918566713323 | Epoch: 576 |

MeanAbsoluteError: 0.0960885882377625 | Loss: 0.0149128609774925 | Epoch: 577 |

MeanAbsoluteError: 0.0977119654417038 | Loss: 0.0166095452788674 | Epoch: 578 |

MeanAbsoluteError: 0.0963776856660843 | Loss: 0.0157931220177367 | Epoch: 579 |

MeanAbsoluteError: 0.1007546558976173 | Loss: 0.0168880955987212 | Epoch: 580 |

MeanAbsoluteError: 0.1005127653479576 | Loss: 0.0159452149858650 | Epoch: 581 |

MeanAbsoluteError: 0.1060678288340569 | Loss: 0.0185537949171461 | Epoch: 582 |

MeanAbsoluteError: 0.0977587923407555 | Loss: 0.0158411980009138 | Epoch: 583 |

MeanAbsoluteError: 0.0975057333707809 | Loss: 0.0159497750048953 | Epoch: 584 |

MeanAbsoluteError: 0.1064710393548012 | Loss: 0.0189864119345051 | Epoch: 585 |

MeanAbsoluteError: 0.0985967069864273 | Loss: 0.0164646540037938 | Epoch: 586 |

MeanAbsoluteError: 0.0978024974465370 | Loss: 0.0165183920611647 | Epoch: 587 |

MeanAbsoluteError: 0.0951704755425453 | Loss: 0.0148399354964689 | Epoch: 588 |

MeanAbsoluteError: 0.1006624698638916 | Loss: 0.0166495151824347 | Epoch: 589 |

MeanAbsoluteError: 0.0985297337174416 | Loss: 0.0157692318865641 | Epoch: 590 |

MeanAbsoluteError: 0.0971502959728241 | Loss: 0.0164393929541075 | Epoch: 591 |

MeanAbsoluteError: 0.0995614007115364 | Loss: 0.0164288475582725 | Epoch: 592 |

MeanAbsoluteError: 0.0996605083346367 | Loss: 0.0166835696795412 | Epoch: 593 |

MeanAbsoluteError: 0.0984433516860008 | Loss: 0.0165848495026876 | Epoch: 594 |

MeanAbsoluteError: 0.0961651876568794 | Loss: 0.0147202890829794 | Epoch: 595 |

MeanAbsoluteError: 0.1008447632193565 | Loss: 0.0166448458551652 | Epoch: 596 |

MeanAbsoluteError: 0.0955660641193390 | Loss: 0.0151021090412663 | Epoch: 597 |

MeanAbsoluteError: 0.0897890329360962 | Loss: 0.0134215467229175 | Epoch: 598 |

MeanAbsoluteError: 0.1018268316984177 | Loss: 0.0172507377310346 | Epoch: 599 |

MeanAbsoluteError: 0.0987390354275703 | Loss: 0.0167196640856370 | Epoch: 600 |

MeanAbsoluteError: 0.0936486274003983 | Loss: 0.0156456941104261 | Epoch: 601 |

MeanAbsoluteError: 0.1000391468405724 | Loss: 0.0161603680091988 | Epoch: 602 |

MeanAbsoluteError: 0.1010711416602135 | Loss: 0.0184721020478173 | Epoch: 603 |

MeanAbsoluteError: 0.0932950899004936 | Loss: 0.0139827936311243 | Epoch: 604 |

MeanAbsoluteError: 0.0963055118918419 | Loss: 0.0153310784988571 | Epoch: 605 |

MeanAbsoluteError: 0.0963570997118950 | Loss: 0.0152647557227950 | Epoch: 606 |

MeanAbsoluteError: 0.0937488079071045 | Loss: 0.0148238821092430 | Epoch: 607 |

MeanAbsoluteError: 0.0985808148980141 | Loss: 0.0164127256039774 | Epoch: 608 |

MeanAbsoluteError: 0.0938173234462738 | Loss: 0.0152247925403450 | Epoch: 609 |

MeanAbsoluteError: 0.0923836380243301 | Loss: 0.0138923459818276 | Epoch: 610 |

MeanAbsoluteError: 0.0898201018571854 | Loss: 0.0137438794830329 | Epoch: 611 |

MeanAbsoluteError: 0.0978201106190681 | Loss: 0.0173729750034787 | Epoch: 612 |

MeanAbsoluteError: 0.0964867696166039 | Loss: 0.0158227476790429 | Epoch: 613 |

MeanAbsoluteError: 0.1026416569948196 | Loss: 0.0170315938178586 | Epoch: 614 |

MeanAbsoluteError: 0.0965701639652252 | Loss: 0.0155963498308847 | Epoch: 615 |

MeanAbsoluteError: 0.0950948297977448 | Loss: 0.0146114285037766 | Epoch: 616 |

MeanAbsoluteError: 0.0904378294944763 | Loss: 0.0143958117053262 | Epoch: 617 |

MeanAbsoluteError: 0.0909558832645416 | Loss: 0.0140302645629466 | Epoch: 618 |

MeanAbsoluteError: 0.0932030603289604 | Loss: 0.0143185231998359 | Epoch: 619 |

MeanAbsoluteError: 0.0962804332375526 | Loss: 0.0159038726518338 | Epoch: 620 |

MeanAbsoluteError: 0.0896650031208992 | Loss: 0.0132685782075714 | Epoch: 621 |

MeanAbsoluteError: 0.0979699566960335 | Loss: 0.0158617348123031 | Epoch: 622 |

MeanAbsoluteError: 0.0942374616861343 | Loss: 0.0145158404502339 | Epoch: 623 |

MeanAbsoluteError: 0.0973250716924667 | Loss: 0.0162320996031485 | Epoch: 624 |

MeanAbsoluteError: 0.0967371091246605 | Loss: 0.0150601549984034 | Epoch: 625 |

MeanAbsoluteError: 0.0968646332621574 | Loss: 0.0153829455666467 | Epoch: 626 |

MeanAbsoluteError: 0.0957232862710953 | Loss: 0.0155192821967406 | Epoch: 627 |

MeanAbsoluteError: 0.0964620858430862 | Loss: 0.0158054238084393 | Epoch: 628 |

MeanAbsoluteError: 0.0965495929121971 | Loss: 0.0160220021387310 | Epoch: 629 |

MeanAbsoluteError: 0.0919289141893387 | Loss: 0.0139654638563904 | Epoch: 630 |

MeanAbsoluteError: 0.0885743051767349 | Loss: 0.0135044062733262 | Epoch: 631 |

MeanAbsoluteError: 0.0953485518693924 | Loss: 0.0147627561067444 | Epoch: 632 |

MeanAbsoluteError: 0.0933807417750359 | Loss: 0.0146393515921955 | Epoch: 633 |

MeanAbsoluteError: 0.0932987630367279 | Loss: 0.0152474323339629 | Epoch: 634 |

MeanAbsoluteError: 0.0967516005039215 | Loss: 0.0151455683295474 | Epoch: 635 |

MeanAbsoluteError: 0.0897047072649002 | Loss: 0.0135201803718034 | Epoch: 636 |

MeanAbsoluteError: 0.0952295288443565 | Loss: 0.0158519333451598 | Epoch: 637 |

MeanAbsoluteError: 0.0909367278218269 | Loss: 0.0144192594697233 | Epoch: 638 |

MeanAbsoluteError: 0.0972539708018303 | Loss: 0.0158647452062966 | Epoch: 639 |

MeanAbsoluteError: 0.1013915613293648 | Loss: 0.0170967591956529 | Epoch: 640 |

MeanAbsoluteError: 0.0947796031832695 | Loss: 0.0150573879413908 | Epoch: 641 |

MeanAbsoluteError: 0.0967868566513062 | Loss: 0.0157545941514642 | Epoch: 642 |

MeanAbsoluteError: 0.0913433134555817 | Loss: 0.0152160621251824 | Epoch: 643 |

MeanAbsoluteError: 0.0961616486310959 | Loss: 0.0149332411096354 | Epoch: 644 |

MeanAbsoluteError: 0.0932204797863960 | Loss: 0.0146495934832880 | Epoch: 645 |

MeanAbsoluteError: 0.0930150002241135 | Loss: 0.0149495071637223 | Epoch: 646 |

MeanAbsoluteError: 0.0900602862238884 | Loss: 0.0134531085098085 | Epoch: 647 |

MeanAbsoluteError: 0.0914531797170639 | Loss: 0.0142307089610646 | Epoch: 648 |

MeanAbsoluteError: 0.0887382775545120 | Loss: 0.0133585218831043 | Epoch: 649 |

MeanAbsoluteError: 0.0921611487865448 | Loss: 0.0139024588831429 | Epoch: 650 |

MeanAbsoluteError: 0.0951155945658684 | Loss: 0.0150498288730644 | Epoch: 651 |

MeanAbsoluteError: 0.0948416292667389 | Loss: 0.0151901409089138 | Epoch: 652 |

MeanAbsoluteError: 0.0924828052520752 | Loss: 0.0149775334747392 | Epoch: 653 |

MeanAbsoluteError: 0.0946136415004730 | Loss: 0.0153754185963044 | Epoch: 654 |

MeanAbsoluteError: 0.0961735174059868 | Loss: 0.0155483722077527 | Epoch: 655 |

MeanAbsoluteError: 0.0998132675886154 | Loss: 0.0167564714831436 | Epoch: 656 |

MeanAbsoluteError: 0.0906401202082634 | Loss: 0.0140924252550091 | Epoch: 657 |

MeanAbsoluteError: 0.0928142890334129 | Loss: 0.0144515420982983 | Epoch: 658 |

MeanAbsoluteError: 0.0917087867856026 | Loss: 0.0137758024128955 | Epoch: 659 |

MeanAbsoluteError: 0.0891851708292961 | Loss: 0.0141792864083739 | Epoch: 660 |

MeanAbsoluteError: 0.0947888344526291 | Loss: 0.0150483565843509 | Epoch: 661 |

MeanAbsoluteError: 0.0943640694022179 | Loss: 0.0145737070912340 | Epoch: 662 |

MeanAbsoluteError: 0.0944503918290138 | Loss: 0.0152485229649271 | Epoch: 663 |

MeanAbsoluteError: 0.0941379368305206 | Loss: 0.0150645027331969 | Epoch: 664 |

MeanAbsoluteError: 0.0872135385870934 | Loss: 0.0127356233672375 | Epoch: 665 |

MeanAbsoluteError: 0.0961293280124664 | Loss: 0.0158609500219851 | Epoch: 666 |

MeanAbsoluteError: 0.0978421568870544 | Loss: 0.0160925480842222 | Epoch: 667 |

MeanAbsoluteError: 0.0916714444756508 | Loss: 0.0139267712061337 | Epoch: 668 |

MeanAbsoluteError: 0.0919057875871658 | Loss: 0.0143144930014842 | Epoch: 669 |

MeanAbsoluteError: 0.0943410843610764 | Loss: 0.0148529598987583 | Epoch: 670 |

MeanAbsoluteError: 0.0938575267791748 | Loss: 0.0154918094991202 | Epoch: 671 |

MeanAbsoluteError: 0.0954762101173401 | Loss: 0.0153640663002686 | Epoch: 672 |

MeanAbsoluteError: 0.0923518836498260 | Loss: 0.0144342011674113 | Epoch: 673 |

MeanAbsoluteError: 0.0838573426008224 | Loss: 0.0130322919080694 | Epoch: 674 |

MeanAbsoluteError: 0.0946438089013100 | Loss: 0.0145477157908802 | Epoch: 675 |

MeanAbsoluteError: 0.0858286917209625 | Loss: 0.0126930823957809 | Epoch: 676 |

MeanAbsoluteError: 0.0893720164895058 | Loss: 0.0135128462814888 | Epoch: 677 |

MeanAbsoluteError: 0.0967947542667389 | Loss: 0.0154100655329228 | Epoch: 678 |

MeanAbsoluteError: 0.0892722830176353 | Loss: 0.0142840799432755 | Epoch: 679 |

MeanAbsoluteError: 0.0921928659081459 | Loss: 0.0144259582498732 | Epoch: 680 |

MeanAbsoluteError: 0.0933601856231689 | Loss: 0.0144121937403300 | Epoch: 681 |

MeanAbsoluteError: 0.0920653566718102 | Loss: 0.0159349769105999 | Epoch: 682 |

MeanAbsoluteError: 0.0949735045433044 | Loss: 0.0150920054744590 | Epoch: 683 |

MeanAbsoluteError: 0.0972508788108826 | Loss: 0.0158260701127438 | Epoch: 684 |

MeanAbsoluteError: 0.0889400914311409 | Loss: 0.0132217998340881 | Epoch: 685 |

MeanAbsoluteError: 0.0918777957558632 | Loss: 0.0146849682164611 | Epoch: 686 |

MeanAbsoluteError: 0.0905194953083992 | Loss: 0.0137007579382771 | Epoch: 687 |

MeanAbsoluteError: 0.0935041978955269 | Loss: 0.0144905740838779 | Epoch: 688 |

MeanAbsoluteError: 0.0894664078950882 | Loss: 0.0133661831833888 | Epoch: 689 |

MeanAbsoluteError: 0.0906479656696320 | Loss: 0.0129881508329102 | Epoch: 690 |

MeanAbsoluteError: 0.0883115604519844 | Loss: 0.0139019429789429 | Epoch: 691 |

MeanAbsoluteError: 0.0943131670355797 | Loss: 0.0146187419280735 | Epoch: 692 |

MeanAbsoluteError: 0.0881499275565147 | Loss: 0.0137641895155214 | Epoch: 693 |

MeanAbsoluteError: 0.0862089097499847 | Loss: 0.0129385813599220 | Epoch: 694 |

MeanAbsoluteError: 0.0862590745091438 | Loss: 0.0130932012956570 | Epoch: 695 |

MeanAbsoluteError: 0.0875169411301613 | Loss: 0.0120731170835400 | Epoch: 696 |

MeanAbsoluteError: 0.0900523662567139 | Loss: 0.0130932060960307 | Epoch: 697 |

MeanAbsoluteError: 0.0929547846317291 | Loss: 0.0153215842312337 | Epoch: 698 |

MeanAbsoluteError: 0.0910495147109032 | Loss: 0.0142463452231217 | Epoch: 699 |

MeanAbsoluteError: 0.0916408002376556 | Loss: 0.0145909430426157 | Epoch: 700 |

MeanAbsoluteError: 0.0909760445356369 | Loss: 0.0144691631801834 | Epoch: 701 |

MeanAbsoluteError: 0.0931015238165855 | Loss: 0.0141941981164079 | Epoch: 702 |

MeanAbsoluteError: 0.0943160578608513 | Loss: 0.0150491890314879 | Epoch: 703 |

MeanAbsoluteError: 0.0883876606822014 | Loss: 0.0133459296058087 | Epoch: 704 |

MeanAbsoluteError: 0.0925025641918182 | Loss: 0.0141147774927958 | Epoch: 705 |

MeanAbsoluteError: 0.0850612819194794 | Loss: 0.0133719610914462 | Epoch: 706 |

MeanAbsoluteError: 0.0900712683796883 | Loss: 0.0133653578141336 | Epoch: 707 |

MeanAbsoluteError: 0.0943104103207588 | Loss: 0.0160831913125003 | Epoch: 708 |

MeanAbsoluteError: 0.0887767896056175 | Loss: 0.0138164001640204 | Epoch: 709 |

MeanAbsoluteError: 0.0884990766644478 | Loss: 0.0131864654082165 | Epoch: 710 |

MeanAbsoluteError: 0.0871781557798386 | Loss: 0.0136024756113450 | Epoch: 711 |

MeanAbsoluteError: 0.0904670208692551 | Loss: 0.0140366526098417 | Epoch: 712 |

MeanAbsoluteError: 0.0845567286014557 | Loss: 0.0122196413710966 | Epoch: 713 |

MeanAbsoluteError: 0.0919441357254982 | Loss: 0.0142646818105519 | Epoch: 714 |

MeanAbsoluteError: 0.0898033604025841 | Loss: 0.0134627972497644 | Epoch: 715 |

MeanAbsoluteError: 0.0952484756708145 | Loss: 0.0146273082156161 | Epoch: 716 |

MeanAbsoluteError: 0.0907735526561737 | Loss: 0.0138906209116006 | Epoch: 717 |

MeanAbsoluteError: 0.0873326137661934 | Loss: 0.0128510094416076 | Epoch: 718 |

MeanAbsoluteError: 0.0867291539907455 | Loss: 0.0134760703152278 | Epoch: 719 |

MeanAbsoluteError: 0.0906604826450348 | Loss: 0.0139623180825341 | Epoch: 720 |

MeanAbsoluteError: 0.0859033614397049 | Loss: 0.0119548634717163 | Epoch: 721 |

MeanAbsoluteError: 0.0863592997193336 | Loss: 0.0119370524008627 | Epoch: 722 |

MeanAbsoluteError: 0.0923669487237930 | Loss: 0.0134630532180502 | Epoch: 723 |

MeanAbsoluteError: 0.0916997194290161 | Loss: 0.0138069968016741 | Epoch: 724 |

MeanAbsoluteError: 0.0852946490049362 | Loss: 0.0125354236539594 | Epoch: 725 |

MeanAbsoluteError: 0.0864159092307091 | Loss: 0.0121496567092133 | Epoch: 726 |

MeanAbsoluteError: 0.0892918780446053 | Loss: 0.0133088386556483 | Epoch: 727 |

MeanAbsoluteError: 0.0892309024930000 | Loss: 0.0138750956342847 | Epoch: 728 |

MeanAbsoluteError: 0.0997823104262352 | Loss: 0.0166005679527492 | Epoch: 729 |

MeanAbsoluteError: 0.0898061096668243 | Loss: 0.0136384960648866 | Epoch: 730 |

MeanAbsoluteError: 0.0943633764982224 | Loss: 0.0139619327787659 | Epoch: 731 |

MeanAbsoluteError: 0.0893125981092453 | Loss: 0.0133629820884986 | Epoch: 732 |

MeanAbsoluteError: 0.0924347862601280 | Loss: 0.0148171652818564 | Epoch: 733 |

MeanAbsoluteError: 0.0907905250787735 | Loss: 0.0137393255522572 | Epoch: 734 |

MeanAbsoluteError: 0.0878291204571724 | Loss: 0.0138885628841429 | Epoch: 735 |

MeanAbsoluteError: 0.0865168571472168 | Loss: 0.0127671975954824 | Epoch: 736 |

MeanAbsoluteError: 0.0880364477634430 | Loss: 0.0129303102962634 | Epoch: 737 |

MeanAbsoluteError: 0.0834802910685539 | Loss: 0.0119109831389142 | Epoch: 738 |

MeanAbsoluteError: 0.0849907025694847 | Loss: 0.0120537501470729 | Epoch: 739 |

MeanAbsoluteError: 0.0839239880442619 | Loss: 0.0127109302923661 | Epoch: 740 |

MeanAbsoluteError: 0.0921679288148880 | Loss: 0.0143589488197661 | Epoch: 741 |

MeanAbsoluteError: 0.0892119109630585 | Loss: 0.0136392097238543 | Epoch: 742 |

MeanAbsoluteError: 0.0920027941465378 | Loss: 0.0141777312652751 | Epoch: 743 |

MeanAbsoluteError: 0.0894734188914299 | Loss: 0.0143954266714475 | Epoch: 744 |

MeanAbsoluteError: 0.0823644250631332 | Loss: 0.0116140862565832 | Epoch: 745 |

MeanAbsoluteError: 0.0886208266019821 | Loss: 0.0130238568094016 | Epoch: 746 |

MeanAbsoluteError: 0.0859970077872276 | Loss: 0.0125071693266121 | Epoch: 747 |

MeanAbsoluteError: 0.0854086875915527 | Loss: 0.0119985524517081 | Epoch: 748 |

MeanAbsoluteError: 0.0916456133127213 | Loss: 0.0146506124286680 | Epoch: 749 |

MeanAbsoluteError: 0.0866899564862251 | Loss: 0.0134203138361772 | Epoch: 750 |

MeanAbsoluteError: 0.0947444215416908 | Loss: 0.0148353840735217 | Epoch: 751 |

MeanAbsoluteError: 0.0897959098219872 | Loss: 0.0142398222726357 | Epoch: 752 |

MeanAbsoluteError: 0.0898419842123985 | Loss: 0.0140729766028865 | Epoch: 753 |

MeanAbsoluteError: 0.0827543437480927 | Loss: 0.0117190304322306 | Epoch: 754 |

MeanAbsoluteError: 0.0864665731787682 | Loss: 0.0124452907306841 | Epoch: 755 |

MeanAbsoluteError: 0.0886324718594551 | Loss: 0.0134533098176083 | Epoch: 756 |

MeanAbsoluteError: 0.0866235569119453 | Loss: 0.0128141998482170 | Epoch: 757 |

MeanAbsoluteError: 0.0884935334324837 | Loss: 0.0127985168645197 | Epoch: 758 |

MeanAbsoluteError: 0.0958149433135986 | Loss: 0.0146632420944904 | Epoch: 759 |

MeanAbsoluteError: 0.0857709422707558 | Loss: 0.0133543467105725 | Epoch: 760 |

MeanAbsoluteError: 0.0862030610442162 | Loss: 0.0119113680015532 | Epoch: 761 |

MeanAbsoluteError: 0.0915297865867615 | Loss: 0.0134914769376822 | Epoch: 762 |

MeanAbsoluteError: 0.0877363383769989 | Loss: 0.0125332345527325 | Epoch: 763 |

MeanAbsoluteError: 0.0861979201436043 | Loss: 0.0126972786899569 | Epoch: 764 |

MeanAbsoluteError: 0.0886425748467445 | Loss: 0.0134177662502043 | Epoch: 765 |

MeanAbsoluteError: 0.0860891193151474 | Loss: 0.0124287321193939 | Epoch: 766 |

MeanAbsoluteError: 0.0865080952644348 | Loss: 0.0133861997649850 | Epoch: 767 |

MeanAbsoluteError: 0.0828128829598427 | Loss: 0.0120354167478217 | Epoch: 768 |

MeanAbsoluteError: 0.0887146294116974 | Loss: 0.0137384025513044 | Epoch: 769 |

MeanAbsoluteError: 0.0918897464871407 | Loss: 0.0138714875392391 | Epoch: 770 |

MeanAbsoluteError: 0.0893371850252151 | Loss: 0.0134091231685791 | Epoch: 771 |

MeanAbsoluteError: 0.0854070559144020 | Loss: 0.0123115474645480 | Epoch: 772 |

MeanAbsoluteError: 0.0887016728520393 | Loss: 0.0138570698190597 | Epoch: 773 |

MeanAbsoluteError: 0.0857271924614906 | Loss: 0.0120408277618117 | Epoch: 774 |

MeanAbsoluteError: 0.0863192081451416 | Loss: 0.0139849752263528 | Epoch: 775 |

MeanAbsoluteError: 0.0906148925423622 | Loss: 0.0145892928799731 | Epoch: 776 |

MeanAbsoluteError: 0.0898269712924957 | Loss: 0.0136065153617134 | Epoch: 777 |

MeanAbsoluteError: 0.0887877866625786 | Loss: 0.0143451428583406 | Epoch: 778 |

MeanAbsoluteError: 0.0886986106634140 | Loss: 0.0136091653279921 | Epoch: 779 |

MeanAbsoluteError: 0.0874193757772446 | Loss: 0.0137000554070012 | Epoch: 780 |

MeanAbsoluteError: 0.0872941836714745 | Loss: 0.0132649398915237 | Epoch: 781 |

MeanAbsoluteError: 0.0920526832342148 | Loss: 0.0144883428317068 | Epoch: 782 |

MeanAbsoluteError: 0.0845841616392136 | Loss: 0.0116353640123998 | Epoch: 783 |

MeanAbsoluteError: 0.0842893421649933 | Loss: 0.0124559426729653 | Epoch: 784 |

MeanAbsoluteError: 0.0844697728753090 | Loss: 0.0122477879914125 | Epoch: 785 |

MeanAbsoluteError: 0.0900035500526428 | Loss: 0.0140212585443290 | Epoch: 786 |

MeanAbsoluteError: 0.0861517786979675 | Loss: 0.0125740609202330 | Epoch: 787 |

MeanAbsoluteError: 0.0916431099176407 | Loss: 0.0139114907555631 | Epoch: 788 |

MeanAbsoluteError: 0.0888753607869148 | Loss: 0.0136382901419430 | Epoch: 789 |

MeanAbsoluteError: 0.0845606774091721 | Loss: 0.0120836158568272 | Epoch: 790 |

MeanAbsoluteError: 0.0807590410113335 | Loss: 0.0114681551585575 | Epoch: 791 |

MeanAbsoluteError: 0.0858226865530014 | Loss: 0.0134262904546146 | Epoch: 792 |

MeanAbsoluteError: 0.0901040732860565 | Loss: 0.0137739601482948 | Epoch: 793 |

MeanAbsoluteError: 0.0905525460839272 | Loss: 0.0143542306405046 | Epoch: 794 |

MeanAbsoluteError: 0.0886818468570709 | Loss: 0.0140806665609125 | Epoch: 795 |

MeanAbsoluteError: 0.0868745073676109 | Loss: 0.0136089015123919 | Epoch: 796 |

MeanAbsoluteError: 0.0862523317337036 | Loss: 0.0132721346324737 | Epoch: 797 |

MeanAbsoluteError: 0.0858037173748016 | Loss: 0.0128083125108969 | Epoch: 798 |

MeanAbsoluteError: 0.0797789767384529 | Loss: 0.0111747501996676 | Epoch: 799 |

MeanAbsoluteError: 0.0864817202091217 | Loss: 0.0125328565467741 | Epoch: 800 |

MeanAbsoluteError: 0.0865729004144669 | Loss: 0.0137278310333689 | Epoch: 801 |

MeanAbsoluteError: 0.0913130342960358 | Loss: 0.0142195880201568 | Epoch: 802 |

MeanAbsoluteError: 0.0816940516233444 | Loss: 0.0110050966604679 | Epoch: 803 |

MeanAbsoluteError: 0.0852877795696259 | Loss: 0.0123793019540123 | Epoch: 804 |

MeanAbsoluteError: 0.0824171900749207 | Loss: 0.0128440501082029 | Epoch: 805 |

MeanAbsoluteError: 0.0907100364565849 | Loss: 0.0140454756035858 | Epoch: 806 |

MeanAbsoluteError: 0.0792649760842323 | Loss: 0.0108858535782444 | Epoch: 807 |

MeanAbsoluteError: 0.0903763696551323 | Loss: 0.0142381561109990 | Epoch: 808 |

MeanAbsoluteError: 0.0842075794935226 | Loss: 0.0125897839139179 | Epoch: 809 |

MeanAbsoluteError: 0.0883259251713753 | Loss: 0.0131113836001411 | Epoch: 810 |

MeanAbsoluteError: 0.0912117362022400 | Loss: 0.0142581266885099 | Epoch: 811 |

MeanAbsoluteError: 0.0814580023288727 | Loss: 0.0111141424582941 | Epoch: 812 |

MeanAbsoluteError: 0.0871970877051353 | Loss: 0.0123103402763991 | Epoch: 813 |

MeanAbsoluteError: 0.0810180529952049 | Loss: 0.0116254900088825 | Epoch: 814 |

MeanAbsoluteError: 0.0811043977737427 | Loss: 0.0115846715714724 | Epoch: 815 |

MeanAbsoluteError: 0.0863296836614609 | Loss: 0.0129724644860350 | Epoch: 816 |

MeanAbsoluteError: 0.0819742679595947 | Loss: 0.0112396295571186 | Epoch: 817 |

MeanAbsoluteError: 0.0805516019463539 | Loss: 0.0116764007932701 | Epoch: 818 |

MeanAbsoluteError: 0.0874178409576416 | Loss: 0.0135617580103766 | Epoch: 819 |

MeanAbsoluteError: 0.0811522752046585 | Loss: 0.0111833336118919 | Epoch: 820 |

MeanAbsoluteError: 0.0804508030414581 | Loss: 0.0117153983263658 | Epoch: 821 |

MeanAbsoluteError: 0.0798240005970001 | Loss: 0.0107536899237554 | Epoch: 822 |

MeanAbsoluteError: 0.0885565280914307 | Loss: 0.0128601583283550 | Epoch: 823 |

MeanAbsoluteError: 0.0884795337915421 | Loss: 0.0134116469078678 | Epoch: 824 |

MeanAbsoluteError: 0.0891752019524574 | Loss: 0.0133455852564415 | Epoch: 825 |

MeanAbsoluteError: 0.0884176790714264 | Loss: 0.0136323052797646 | Epoch: 826 |

MeanAbsoluteError: 0.0902144834399223 | Loss: 0.0137431452028492 | Epoch: 827 |

MeanAbsoluteError: 0.0853938832879066 | Loss: 0.0125286858970200 | Epoch: 828 |

MeanAbsoluteError: 0.0883293449878693 | Loss: 0.0128368641608783 | Epoch: 829 |

MeanAbsoluteError: 0.0809650570154190 | Loss: 0.0122426609583514 | Epoch: 830 |

MeanAbsoluteError: 0.0836098641157150 | Loss: 0.0124808507520356 | Epoch: 831 |

MeanAbsoluteError: 0.0853480845689774 | Loss: 0.0122655625230982 | Epoch: 832 |

MeanAbsoluteError: 0.0859313681721687 | Loss: 0.0130010988933403 | Epoch: 833 |

MeanAbsoluteError: 0.0836371183395386 | Loss: 0.0121528218423070 | Epoch: 834 |

MeanAbsoluteError: 0.0804404541850090 | Loss: 0.0110012058395174 | Epoch: 835 |

MeanAbsoluteError: 0.0875793993473053 | Loss: 0.0129802084630986 | Epoch: 836 |

MeanAbsoluteError: 0.0856867060065269 | Loss: 0.0127158584193967 | Epoch: 837 |

MeanAbsoluteError: 0.0867143124341965 | Loss: 0.0127852808585158 | Epoch: 838 |

MeanAbsoluteError: 0.0828242823481560 | Loss: 0.0125470602643327 | Epoch: 839 |

MeanAbsoluteError: 0.0862727910280228 | Loss: 0.0123975050611140 | Epoch: 840 |

MeanAbsoluteError: 0.0864468663930893 | Loss: 0.0122420395240382 | Epoch: 841 |

MeanAbsoluteError: 0.0861263945698738 | Loss: 0.0121274420667517 | Epoch: 842 |

MeanAbsoluteError: 0.0813935697078705 | Loss: 0.0115135153244773 | Epoch: 843 |

MeanAbsoluteError: 0.0868164226412773 | Loss: 0.0129945761853257 | Epoch: 844 |

MeanAbsoluteError: 0.0806476250290871 | Loss: 0.0118165346844277 | Epoch: 845 |

MeanAbsoluteError: 0.0831377282738686 | Loss: 0.0114626422768924 | Epoch: 846 |

MeanAbsoluteError: 0.0831957608461380 | Loss: 0.0120046072629945 | Epoch: 847 |

MeanAbsoluteError: 0.0885964557528496 | Loss: 0.0140025000152491 | Epoch: 848 |

MeanAbsoluteError: 0.0845857113599777 | Loss: 0.0130284672452884 | Epoch: 849 |

MeanAbsoluteError: 0.0861662924289703 | Loss: 0.0123421718223835 | Epoch: 850 |

MeanAbsoluteError: 0.0839953944087029 | Loss: 0.0120646963269246 | Epoch: 851 |

MeanAbsoluteError: 0.0852303877472878 | Loss: 0.0119981294839575 | Epoch: 852 |

MeanAbsoluteError: 0.0844116136431694 | Loss: 0.0120764471249034 | Epoch: 853 |

MeanAbsoluteError: 0.0827642083168030 | Loss: 0.0115452601076686 | Epoch: 854 |

MeanAbsoluteError: 0.0796970427036285 | Loss: 0.0112214653990668 | Epoch: 855 |

MeanAbsoluteError: 0.0821976289153099 | Loss: 0.0112331208280132 | Epoch: 856 |

MeanAbsoluteError: 0.0822429135441780 | Loss: 0.0119439265374967 | Epoch: 857 |

MeanAbsoluteError: 0.0814012065529823 | Loss: 0.0117733437980011 | Epoch: 858 |

MeanAbsoluteError: 0.0856880024075508 | Loss: 0.0129521806114644 | Epoch: 859 |

MeanAbsoluteError: 0.0763607844710350 | Loss: 0.0094642176222836 | Epoch: 860 |

MeanAbsoluteError: 0.0835796818137169 | Loss: 0.0122410474442101 | Epoch: 861 |

MeanAbsoluteError: 0.0869519263505936 | Loss: 0.0132485300824207 | Epoch: 862 |

MeanAbsoluteError: 0.0832964107394218 | Loss: 0.0124433065639308 | Epoch: 863 |

MeanAbsoluteError: 0.0847704187035561 | Loss: 0.0125383529395428 | Epoch: 864 |

MeanAbsoluteError: 0.0793285369873047 | Loss: 0.0108464179500273 | Epoch: 865 |

MeanAbsoluteError: 0.0781135037541389 | Loss: 0.0099608369560823 | Epoch: 866 |

MeanAbsoluteError: 0.0753740370273590 | Loss: 0.0094252882721412 | Epoch: 867 |

MeanAbsoluteError: 0.0779273733496666 | Loss: 0.0110048592695966 | Epoch: 868 |

MeanAbsoluteError: 0.0815459415316582 | Loss: 0.0125639568947766 | Epoch: 869 |

MeanAbsoluteError: 0.0825504884123802 | Loss: 0.0124895045197506 | Epoch: 870 |

MeanAbsoluteError: 0.0781207233667374 | Loss: 0.0103462629565305 | Epoch: 871 |

MeanAbsoluteError: 0.0805160403251648 | Loss: 0.0117348891064466 | Epoch: 872 |

MeanAbsoluteError: 0.0860563963651657 | Loss: 0.0122281214998414 | Epoch: 873 |

MeanAbsoluteError: 0.0775562003254890 | Loss: 0.0105889061064227 | Epoch: 874 |

MeanAbsoluteError: 0.0903373062610626 | Loss: 0.0133733561292562 | Epoch: 875 |

MeanAbsoluteError: 0.0858713462948799 | Loss: 0.0127553045549818 | Epoch: 876 |

MeanAbsoluteError: 0.0857159346342087 | Loss: 0.0118936768372805 | Epoch: 877 |

MeanAbsoluteError: 0.0869218185544014 | Loss: 0.0133481917849955 | Epoch: 878 |

MeanAbsoluteError: 0.0793927162885666 | Loss: 0.0116182592293747 | Epoch: 879 |

MeanAbsoluteError: 0.0816598013043404 | Loss: 0.0120777431310853 | Epoch: 880 |

MeanAbsoluteError: 0.0865908265113831 | Loss: 0.0121942995845651 | Epoch: 881 |

MeanAbsoluteError: 0.0847345292568207 | Loss: 0.0117499036064328 | Epoch: 882 |

MeanAbsoluteError: 0.0848828107118607 | Loss: 0.0115949919461430 | Epoch: 883 |

MeanAbsoluteError: 0.0828242152929306 | Loss: 0.0123472943096810 | Epoch: 884 |

MeanAbsoluteError: 0.0823848471045494 | Loss: 0.0119550356885884 | Epoch: 885 |

MeanAbsoluteError: 0.0861052870750427 | Loss: 0.0133195743826703 | Epoch: 886 |

MeanAbsoluteError: 0.0837353989481926 | Loss: 0.0130225233925739 | Epoch: 887 |

MeanAbsoluteError: 0.0817463696002960 | Loss: 0.0124399826331743 | Epoch: 888 |

MeanAbsoluteError: 0.0820957645773888 | Loss: 0.0117472662106350 | Epoch: 889 |

MeanAbsoluteError: 0.0816173478960991 | Loss: 0.0118197288089626 | Epoch: 890 |

MeanAbsoluteError: 0.0764997899532318 | Loss: 0.0106816185817903 | Epoch: 891 |

MeanAbsoluteError: 0.0771630704402924 | Loss: 0.0100602713887444 | Epoch: 892 |

MeanAbsoluteError: 0.0765248462557793 | Loss: 0.0109205758494015 | Epoch: 893 |

MeanAbsoluteError: 0.0872589051723480 | Loss: 0.0135042275157563 | Epoch: 894 |

MeanAbsoluteError: 0.0830823704600334 | Loss: 0.0118426920597024 | Epoch: 895 |

MeanAbsoluteError: 0.0817200094461441 | Loss: 0.0123003015637520 | Epoch: 896 |

MeanAbsoluteError: 0.0787248834967613 | Loss: 0.0112075292940911 | Epoch: 897 |

MeanAbsoluteError: 0.0816954150795937 | Loss: 0.0112692587218771 | Epoch: 898 |

MeanAbsoluteError: 0.0812734365463257 | Loss: 0.0118055330875116 | Epoch: 899 |

MeanAbsoluteError: 0.0832881331443787 | Loss: 0.0123559036572503 | Epoch: 900 |

MeanAbsoluteError: 0.0865415483713150 | Loss: 0.0128937351900580 | Epoch: 901 |

MeanAbsoluteError: 0.0795857906341553 | Loss: 0.0110751351706328 | Epoch: 902 |

MeanAbsoluteError: 0.0871486440300941 | Loss: 0.0132788756764664 | Epoch: 903 |

MeanAbsoluteError: 0.0812246352434158 | Loss: 0.0122140400755840 | Epoch: 904 |

MeanAbsoluteError: 0.0838865488767624 | Loss: 0.0126089679376976 | Epoch: 905 |

MeanAbsoluteError: 0.0797151103615761 | Loss: 0.0114307044782618 | Epoch: 906 |

MeanAbsoluteError: 0.0863653719425201 | Loss: 0.0134116959610158 | Epoch: 907 |

MeanAbsoluteError: 0.0768132954835892 | Loss: 0.0101878985298875 | Epoch: 908 |

MeanAbsoluteError: 0.0830380693078041 | Loss: 0.0123688119220121 | Epoch: 909 |

MeanAbsoluteError: 0.0826868712902069 | Loss: 0.0116514513273554 | Epoch: 910 |

MeanAbsoluteError: 0.0786157175898552 | Loss: 0.0103059144022457 | Epoch: 911 |

MeanAbsoluteError: 0.0830175131559372 | Loss: 0.0127984374776353 | Epoch: 912 |

MeanAbsoluteError: 0.0856058821082115 | Loss: 0.0126033040619222 | Epoch: 913 |

MeanAbsoluteError: 0.0804712772369385 | Loss: 0.0104722084886938 | Epoch: 914 |

MeanAbsoluteError: 0.0815507322549820 | Loss: 0.0122549755098286 | Epoch: 915 |

MeanAbsoluteError: 0.0802043378353119 | Loss: 0.0109584867674615 | Epoch: 916 |

MeanAbsoluteError: 0.0799411013722420 | Loss: 0.0111678197145109 | Epoch: 917 |

MeanAbsoluteError: 0.0833179950714111 | Loss: 0.0118655503335322 | Epoch: 918 |

MeanAbsoluteError: 0.0823610499501228 | Loss: 0.0116880614956911 | Epoch: 919 |

MeanAbsoluteError: 0.0783708989620209 | Loss: 0.0102864640241023 | Epoch: 920 |

MeanAbsoluteError: 0.0803677812218666 | Loss: 0.0111024491784580 | Epoch: 921 |

MeanAbsoluteError: 0.0784214362502098 | Loss: 0.0111703036696902 | Epoch: 922 |

MeanAbsoluteError: 0.0830733329057693 | Loss: 0.0123479970693006 | Epoch: 923 |

MeanAbsoluteError: 0.0800443440675735 | Loss: 0.0115271864702421 | Epoch: 924 |

MeanAbsoluteError: 0.0815910920500755 | Loss: 0.0116231403778268 | Epoch: 925 |

MeanAbsoluteError: 0.0807124823331833 | Loss: 0.0114453103237611 | Epoch: 926 |

MeanAbsoluteError: 0.0814790353178978 | Loss: 0.0118356360201627 | Epoch: 927 |

MeanAbsoluteError: 0.0767167061567307 | Loss: 0.0101332863705708 | Epoch: 928 |

MeanAbsoluteError: 0.0738124027848244 | Loss: 0.0093464799875558 | Epoch: 929 |

MeanAbsoluteError: 0.0843003392219543 | Loss: 0.0118630059461187 | Epoch: 930 |

MeanAbsoluteError: 0.0799879357218742 | Loss: 0.0109510113059014 | Epoch: 931 |

MeanAbsoluteError: 0.0842466950416565 | Loss: 0.0123554844253037 | Epoch: 932 |

MeanAbsoluteError: 0.0848404914140701 | Loss: 0.0122660007665399 | Epoch: 933 |

MeanAbsoluteError: 0.0807810202240944 | Loss: 0.0112789852743542 | Epoch: 934 |

MeanAbsoluteError: 0.0828318893909454 | Loss: 0.0116049692504748 | Epoch: 935 |

MeanAbsoluteError: 0.0737516954541206 | Loss: 0.0094814870978735 | Epoch: 936 |

MeanAbsoluteError: 0.0825454816222191 | Loss: 0.0116487422356658 | Epoch: 937 |

MeanAbsoluteError: 0.0815281942486763 | Loss: 0.0114970910390366 | Epoch: 938 |

MeanAbsoluteError: 0.0775657072663307 | Loss: 0.0107953030587426 | Epoch: 939 |

MeanAbsoluteError: 0.0796855166554451 | Loss: 0.0106407967192111 | Epoch: 940 |

MeanAbsoluteError: 0.0819826051592827 | Loss: 0.0110130202765868 | Epoch: 941 |

MeanAbsoluteError: 0.0792942345142365 | Loss: 0.0105002150441336 | Epoch: 942 |

MeanAbsoluteError: 0.0851732641458511 | Loss: 0.0116267241741055 | Epoch: 943 |

MeanAbsoluteError: 0.0839510932564735 | Loss: 0.0119949825932781 | Epoch: 944 |

MeanAbsoluteError: 0.0774140581488609 | Loss: 0.0101660285940306 | Epoch: 945 |

MeanAbsoluteError: 0.0802039578557014 | Loss: 0.0115133943254477 | Epoch: 946 |

MeanAbsoluteError: 0.0767598897218704 | Loss: 0.0105067433767060 | Epoch: 947 |

MeanAbsoluteError: 0.0784007534384727 | Loss: 0.0102688882080110 | Epoch: 948 |

MeanAbsoluteError: 0.0820533335208893 | Loss: 0.0120548734272597 | Epoch: 949 |

MeanAbsoluteError: 0.0806204676628113 | Loss: 0.0119157742782651 | Epoch: 950 |

MeanAbsoluteError: 0.0797103717923164 | Loss: 0.0123256436613641 | Epoch: 951 |

MeanAbsoluteError: 0.0741716623306274 | Loss: 0.0104491677812287 | Epoch: 952 |

MeanAbsoluteError: 0.0815588459372520 | Loss: 0.0110812778642624 | Epoch: 953 |

MeanAbsoluteError: 0.0787247046828270 | Loss: 0.0102688081565672 | Epoch: 954 |

MeanAbsoluteError: 0.0823978334665298 | Loss: 0.0115711442616278 | Epoch: 955 |

MeanAbsoluteError: 0.0756896957755089 | Loss: 0.0100838043547265 | Epoch: 956 |

MeanAbsoluteError: 0.0786683708429337 | Loss: 0.0118535859357507 | Epoch: 957 |

MeanAbsoluteError: 0.0837032645940781 | Loss: 0.0118028071187534 | Epoch: 958 |

MeanAbsoluteError: 0.0759436413645744 | Loss: 0.0114929044958899 | Epoch: 959 |

MeanAbsoluteError: 0.0854881703853607 | Loss: 0.0124227013853185 | Epoch: 960 |

MeanAbsoluteError: 0.0764975547790527 | Loss: 0.0108291890088731 | Epoch: 961 |

MeanAbsoluteError: 0.0818009600043297 | Loss: 0.0118468947408837 | Epoch: 962 |

MeanAbsoluteError: 0.0814691632986069 | Loss: 0.0111229849141091 | Epoch: 963 |

MeanAbsoluteError: 0.0788517817854881 | Loss: 0.0112426976535971 | Epoch: 964 |

MeanAbsoluteError: 0.0808836519718170 | Loss: 0.0115413672662301 | Epoch: 965 |

MeanAbsoluteError: 0.0782296583056450 | Loss: 0.0108235859355773 | Epoch: 966 |

MeanAbsoluteError: 0.0745306238532066 | Loss: 0.0103440213458574 | Epoch: 967 |

MeanAbsoluteError: 0.0774982497096062 | Loss: 0.0107743175597473 | Epoch: 968 |

MeanAbsoluteError: 0.0763395875692368 | Loss: 0.0098566429838926 | Epoch: 969 |

MeanAbsoluteError: 0.0828440785408020 | Loss: 0.0119414900919401 | Epoch: 970 |

MeanAbsoluteError: 0.0729564279317856 | Loss: 0.0091701747135327 | Epoch: 971 |

MeanAbsoluteError: 0.0743482485413551 | Loss: 0.0100573152975873 | Epoch: 972 |

MeanAbsoluteError: 0.0762059167027473 | Loss: 0.0102828313098871 | Epoch: 973 |

MeanAbsoluteError: 0.0823271721601486 | Loss: 0.0123332749994006 | Epoch: 974 |

MeanAbsoluteError: 0.0769732221961021 | Loss: 0.0099090052180691 | Epoch: 975 |

MeanAbsoluteError: 0.0755265876650810 | Loss: 0.0106612317410084 | Epoch: 976 |

MeanAbsoluteError: 0.0780579149723053 | Loss: 0.0109656291760014 | Epoch: 977 |

MeanAbsoluteError: 0.0729009881615639 | Loss: 0.0097179506270671 | Epoch: 978 |

MeanAbsoluteError: 0.0814624130725861 | Loss: 0.0122003005626902 | Epoch: 979 |

MeanAbsoluteError: 0.0756580233573914 | Loss: 0.0101574591720419 | Epoch: 980 |

MeanAbsoluteError: 0.0772022828459740 | Loss: 0.0105531932566646 | Epoch: 981 |

MeanAbsoluteError: 0.0786762461066246 | Loss: 0.0122674369444333 | Epoch: 982 |

MeanAbsoluteError: 0.0732318088412285 | Loss: 0.0088566838993574 | Epoch: 983 |

MeanAbsoluteError: 0.0825566425919533 | Loss: 0.0124756342571345 | Epoch: 984 |

MeanAbsoluteError: 0.0766626074910164 | Loss: 0.0109638466881491 | Epoch: 985 |

MeanAbsoluteError: 0.0773973613977432 | Loss: 0.0103710649581141 | Epoch: 986 |

MeanAbsoluteError: 0.0831999331712723 | Loss: 0.0115988148086277 | Epoch: 987 |

MeanAbsoluteError: 0.0765479952096939 | Loss: 0.0098583799475212 | Epoch: 988 |

MeanAbsoluteError: 0.0755728185176849 | Loss: 0.0094789150768338 | Epoch: 989 |

MeanAbsoluteError: 0.0752402916550636 | Loss: 0.0106695184342000 | Epoch: 990 |

MeanAbsoluteError: 0.0797457918524742 | Loss: 0.0119888560156141 | Epoch: 991 |

MeanAbsoluteError: 0.0863485187292099 | Loss: 0.0133733683711519 | Epoch: 992 |

MeanAbsoluteError: 0.0731573179364204 | Loss: 0.0099998498571707 | Epoch: 993 |

MeanAbsoluteError: 0.0762025415897369 | Loss: 0.0106176555088314 | Epoch: 994 |

MeanAbsoluteError: 0.0705077648162842 | Loss: 0.0096937311786557 | Epoch: 995 |

MeanAbsoluteError: 0.0733729973435402 | Loss: 0.0095583695450356 | Epoch: 996 |

MeanAbsoluteError: 0.0831533744931221 | Loss: 0.0116547464509222 | Epoch: 997 |

MeanAbsoluteError: 0.0781236514449120 | Loss: 0.0115017312044256 | Epoch: 998 |

MeanAbsoluteError: 0.0746921896934509 | Loss: 0.0098833642837296 | Epoch: 999 |

MeanAbsoluteError: 0.0753914043307304 | Loss: 0.0103633458494642 | Epoch: 1000 |

MeanAbsoluteError: 0.0757559761404991 | Loss: 0.0097833661587841 | Epoch: 1001 |

MeanAbsoluteError: 0.0705480352044106 | Loss: 0.0085188920716852 | Epoch: 1002 |

MeanAbsoluteError: 0.0768457427620888 | Loss: 0.0099748401825127 | Epoch: 1003 |

MeanAbsoluteError: 0.0831488072872162 | Loss: 0.0120860961836297 | Epoch: 1004 |

MeanAbsoluteError: 0.0798242017626762 | Loss: 0.0103526326721961 | Epoch: 1005 |

MeanAbsoluteError: 0.0789614543318748 | Loss: 0.0107964140716892 | Epoch: 1006 |

MeanAbsoluteError: 0.0754928812384605 | Loss: 0.0103014436764837 | Epoch: 1007 |

MeanAbsoluteError: 0.0756279230117798 | Loss: 0.0106289008436336 | Epoch: 1008 |

MeanAbsoluteError: 0.0778771042823792 | Loss: 0.0117219726176942 | Epoch: 1009 |

MeanAbsoluteError: 0.0777323171496391 | Loss: 0.0110546496565318 | Epoch: 1010 |

MeanAbsoluteError: 0.0744861587882042 | Loss: 0.0104507397101891 | Epoch: 1011 |

MeanAbsoluteError: 0.0722341686487198 | Loss: 0.0090870050431598 | Epoch: 1012 |

MeanAbsoluteError: 0.0743695795536041 | Loss: 0.0096118649369358 | Epoch: 1013 |

MeanAbsoluteError: 0.0788989737629890 | Loss: 0.0109064492666706 | Epoch: 1014 |

MeanAbsoluteError: 0.0738049969077110 | Loss: 0.0096501302412071 | Epoch: 1015 |

MeanAbsoluteError: 0.0783030539751053 | Loss: 0.0111113049119012 | Epoch: 1016 |

MeanAbsoluteError: 0.0739242061972618 | Loss: 0.0098138897991642 | Epoch: 1017 |

MeanAbsoluteError: 0.0779002010822296 | Loss: 0.0104692062443367 | Epoch: 1018 |

MeanAbsoluteError: 0.0783542543649673 | Loss: 0.0107558503670346 | Epoch: 1019 |

MeanAbsoluteError: 0.0818097591400146 | Loss: 0.0110196580595160 | Epoch: 1020 |

MeanAbsoluteError: 0.0738707855343819 | Loss: 0.0095344771980551 | Epoch: 1021 |

MeanAbsoluteError: 0.0776766240596771 | Loss: 0.0110936028573936 | Epoch: 1022 |

MeanAbsoluteError: 0.0834272205829620 | Loss: 0.0119343298724683 | Epoch: 1023 |

MeanAbsoluteError: 0.0767625719308853 | Loss: 0.0115287624870931 | Epoch: 1024 |

MeanAbsoluteError: 0.0751731842756271 | Loss: 0.0100555483445108 | Epoch: 1025 |

MeanAbsoluteError: 0.0811874493956566 | Loss: 0.0119457913245424 | Epoch: 1026 |

MeanAbsoluteError: 0.0766018852591515 | Loss: 0.0109401474932383 | Epoch: 1027 |

MeanAbsoluteError: 0.0769082084298134 | Loss: 0.0105266738980148 | Epoch: 1028 |

MeanAbsoluteError: 0.0779322609305382 | Loss: 0.0113678245340028 | Epoch: 1029 |

MeanAbsoluteError: 0.0764741227030754 | Loss: 0.0107084866911221 | Epoch: 1030 |

MeanAbsoluteError: 0.0770469531416893 | Loss: 0.0110057342163054 | Epoch: 1031 |

MeanAbsoluteError: 0.0770557820796967 | Loss: 0.0108231611666997 | Epoch: 1032 |

MeanAbsoluteError: 0.0788829773664474 | Loss: 0.0109902932136417 | Epoch: 1033 |

MeanAbsoluteError: 0.0760050937533379 | Loss: 0.0105878888305354 | Epoch: 1034 |

MeanAbsoluteError: 0.0756910294294357 | Loss: 0.0106224919659144 | Epoch: 1035 |

MeanAbsoluteError: 0.0786144360899925 | Loss: 0.0112265149529986 | Epoch: 1036 |

MeanAbsoluteError: 0.0741284340620041 | Loss: 0.0095825079614588 | Epoch: 1037 |

MeanAbsoluteError: 0.0775757506489754 | Loss: 0.0108294108243960 | Epoch: 1038 |

MeanAbsoluteError: 0.0767244920134544 | Loss: 0.0103126389273605 | Epoch: 1039 |

MeanAbsoluteError: 0.0801506116986275 | Loss: 0.0117465250859580 | Epoch: 1040 |

MeanAbsoluteError: 0.0746002569794655 | Loss: 0.0096707050510668 | Epoch: 1041 |

MeanAbsoluteError: 0.0799166709184647 | Loss: 0.0117579320412915 | Epoch: 1042 |

MeanAbsoluteError: 0.0715933367609978 | Loss: 0.0090493200034325 | Epoch: 1043 |

MeanAbsoluteError: 0.0739725902676582 | Loss: 0.0092101480209385 | Epoch: 1044 |

MeanAbsoluteError: 0.0723491311073303 | Loss: 0.0090113187919633 | Epoch: 1045 |

MeanAbsoluteError: 0.0780948251485825 | Loss: 0.0109670229688951 | Epoch: 1046 |

MeanAbsoluteError: 0.0728547275066376 | Loss: 0.0094306538352976 | Epoch: 1047 |

MeanAbsoluteError: 0.0805683657526970 | Loss: 0.0122495400344390 | Epoch: 1048 |

MeanAbsoluteError: 0.0740289092063904 | Loss: 0.0089177032314425 | Epoch: 1049 |

MeanAbsoluteError: 0.0705036520957947 | Loss: 0.0091240585041426 | Epoch: 1050 |

MeanAbsoluteError: 0.0773598030209541 | Loss: 0.0106342341379786 | Epoch: 1051 |

MeanAbsoluteError: 0.0714877992868423 | Loss: 0.0095687410289270 | Epoch: 1052 |

MeanAbsoluteError: 0.0746006220579147 | Loss: 0.0101405670919363 | Epoch: 1053 |

MeanAbsoluteError: 0.0692896693944931 | Loss: 0.0090205796544372 | Epoch: 1054 |

MeanAbsoluteError: 0.0780404880642891 | Loss: 0.0102410355265602 | Epoch: 1055 |

MeanAbsoluteError: 0.0682125538587570 | Loss: 0.0091718223462037 | Epoch: 1056 |

MeanAbsoluteError: 0.0713212937116623 | Loss: 0.0090043592757987 | Epoch: 1057 |

MeanAbsoluteError: 0.0743348598480225 | Loss: 0.0097596993275026 | Epoch: 1058 |

MeanAbsoluteError: 0.0771271139383316 | Loss: 0.0108261842059134 | Epoch: 1059 |

MeanAbsoluteError: 0.0721574947237968 | Loss: 0.0093124164586031 | Epoch: 1060 |

MeanAbsoluteError: 0.0775763094425201 | Loss: 0.0108752375261004 | Epoch: 1061 |

MeanAbsoluteError: 0.0757887810468674 | Loss: 0.0098738333700961 | Epoch: 1062 |

MeanAbsoluteError: 0.0788694992661476 | Loss: 0.0102308078307397 | Epoch: 1063 |

MeanAbsoluteError: 0.0761614963412285 | Loss: 0.0104665569261548 | Epoch: 1064 |

MeanAbsoluteError: 0.0738383457064629 | Loss: 0.0094858589958433 | Epoch: 1065 |

MeanAbsoluteError: 0.0744706764817238 | Loss: 0.0105799999288502 | Epoch: 1066 |

MeanAbsoluteError: 0.0748436897993088 | Loss: 0.0096650467131985 | Epoch: 1067 |

MeanAbsoluteError: 0.0832228660583496 | Loss: 0.0120505646681462 | Epoch: 1068 |

MeanAbsoluteError: 0.0707968398928642 | Loss: 0.0098071922517556 | Epoch: 1069 |

MeanAbsoluteError: 0.0693565458059311 | Loss: 0.0085283380734957 | Epoch: 1070 |

MeanAbsoluteError: 0.0785540416836739 | Loss: 0.0112147453500074 | Epoch: 1071 |

MeanAbsoluteError: 0.0748221576213837 | Loss: 0.0110112781646482 | Epoch: 1072 |

MeanAbsoluteError: 0.0756680071353912 | Loss: 0.0105041665949102 | Epoch: 1073 |

MeanAbsoluteError: 0.0734638497233391 | Loss: 0.0102985271880364 | Epoch: 1074 |

MeanAbsoluteError: 0.0750446915626526 | Loss: 0.0101232184694163 | Epoch: 1075 |

MeanAbsoluteError: 0.0746998935937881 | Loss: 0.0104511256028976 | Epoch: 1076 |

MeanAbsoluteError: 0.0709458291530609 | Loss: 0.0096882451467536 | Epoch: 1077 |

MeanAbsoluteError: 0.0789560005068779 | Loss: 0.0110676009262534 | Epoch: 1078 |

MeanAbsoluteError: 0.0753116086125374 | Loss: 0.0100056998500077 | Epoch: 1079 |

MeanAbsoluteError: 0.0702742412686348 | Loss: 0.0086896320552720 | Epoch: 1080 |

MeanAbsoluteError: 0.0747972279787064 | Loss: 0.0108010869031811 | Epoch: 1081 |

MeanAbsoluteError: 0.0779358074069023 | Loss: 0.0103702746947723 | Epoch: 1082 |

MeanAbsoluteError: 0.0780492499470711 | Loss: 0.0103656680810673 | Epoch: 1083 |

MeanAbsoluteError: 0.0772058516740799 | Loss: 0.0103615869295754 | Epoch: 1084 |

MeanAbsoluteError: 0.0773868337273598 | Loss: 0.0105664778420517 | Epoch: 1085 |

MeanAbsoluteError: 0.0762199908494949 | Loss: 0.0098720195608621 | Epoch: 1086 |

MeanAbsoluteError: 0.0727044865489006 | Loss: 0.0093265218216250 | Epoch: 1087 |

MeanAbsoluteError: 0.0696963146328926 | Loss: 0.0077078896605841 | Epoch: 1088 |

MeanAbsoluteError: 0.0761327520012856 | Loss: 0.0104083669125976 | Epoch: 1089 |

MeanAbsoluteError: 0.0761842504143715 | Loss: 0.0106975709909830 | Epoch: 1090 |

MeanAbsoluteError: 0.0730719640851021 | Loss: 0.0096656522855604 | Epoch: 1091 |

MeanAbsoluteError: 0.0738746300339699 | Loss: 0.0099440617063859 | Epoch: 1092 |

MeanAbsoluteError: 0.0782159715890884 | Loss: 0.0108302825904684 | Epoch: 1093 |

MeanAbsoluteError: 0.0703880786895752 | Loss: 0.0094506084373036 | Epoch: 1094 |

MeanAbsoluteError: 0.0761565789580345 | Loss: 0.0101296130256742 | Epoch: 1095 |

MeanAbsoluteError: 0.0725969597697258 | Loss: 0.0089528841100885 | Epoch: 1096 |

MeanAbsoluteError: 0.0728861168026924 | Loss: 0.0100698394920619 | Epoch: 1097 |

MeanAbsoluteError: 0.0786394998431206 | Loss: 0.0109851752032894 | Epoch: 1098 |

MeanAbsoluteError: 0.0706152990460396 | Loss: 0.0087707446792774 | Epoch: 1099 |

MeanAbsoluteError: 0.0716203525662422 | Loss: 0.0092158347686442 | Epoch: 1100 |

MeanAbsoluteError: 0.0742752701044083 | Loss: 0.0098511454547164 | Epoch: 1101 |

MeanAbsoluteError: 0.0761006921529770 | Loss: 0.0099675892440185 | Epoch: 1102 |

MeanAbsoluteError: 0.0725260078907013 | Loss: 0.0097420433982794 | Epoch: 1103 |

MeanAbsoluteError: 0.0724799707531929 | Loss: 0.0098004627591945 | Epoch: 1104 |

MeanAbsoluteError: 0.0708270221948624 | Loss: 0.0094381833554265 | Epoch: 1105 |

MeanAbsoluteError: 0.0766417309641838 | Loss: 0.0110089998758485 | Epoch: 1106 |

MeanAbsoluteError: 0.0707125589251518 | Loss: 0.0087864652068068 | Epoch: 1107 |

MeanAbsoluteError: 0.0748772174119949 | Loss: 0.0105487041123464 | Epoch: 1108 |

MeanAbsoluteError: 0.0771226212382317 | Loss: 0.0106066883045666 | Epoch: 1109 |

MeanAbsoluteError: 0.0733430758118629 | Loss: 0.0100652965901948 | Epoch: 1110 |

MeanAbsoluteError: 0.0707973763346672 | Loss: 0.0088674505786306 | Epoch: 1111 |

MeanAbsoluteError: 0.0780311599373817 | Loss: 0.0108499055122714 | Epoch: 1112 |

MeanAbsoluteError: 0.0753980353474617 | Loss: 0.0102129101518464 | Epoch: 1113 |

MeanAbsoluteError: 0.0754498988389969 | Loss: 0.0097423461076687 | Epoch: 1114 |

MeanAbsoluteError: 0.0739755332469940 | Loss: 0.0100067125013690 | Epoch: 1115 |

MeanAbsoluteError: 0.0680846497416496 | Loss: 0.0083677069948559 | Epoch: 1116 |

MeanAbsoluteError: 0.0724154114723206 | Loss: 0.0092706432281799 | Epoch: 1117 |

MeanAbsoluteError: 0.0740863382816315 | Loss: 0.0099931320892260 | Epoch: 1118 |

MeanAbsoluteError: 0.0734381452202797 | Loss: 0.0096612278298987 | Epoch: 1119 |

MeanAbsoluteError: 0.0686213746666908 | Loss: 0.0085499803735418 | Epoch: 1120 |

MeanAbsoluteError: 0.0715530365705490 | Loss: 0.0093417449182743 | Epoch: 1121 |

MeanAbsoluteError: 0.0736451894044876 | Loss: 0.0094220094171760 | Epoch: 1122 |

MeanAbsoluteError: 0.0738243982195854 | Loss: 0.0092302234060965 | Epoch: 1123 |

MeanAbsoluteError: 0.0718576461076736 | Loss: 0.0092554958019726 | Epoch: 1124 |

MeanAbsoluteError: 0.0770840942859650 | Loss: 0.0112132324319809 | Epoch: 1125 |

MeanAbsoluteError: 0.0735283121466637 | Loss: 0.0101361741174727 | Epoch: 1126 |

MeanAbsoluteError: 0.0713175982236862 | Loss: 0.0089035067124011 | Epoch: 1127 |

MeanAbsoluteError: 0.0743960216641426 | Loss: 0.0099704417186634 | Epoch: 1128 |

MeanAbsoluteError: 0.0750558301806450 | Loss: 0.0098137166045975 | Epoch: 1129 |

MeanAbsoluteError: 0.0748309418559074 | Loss: 0.0094900925181355 | Epoch: 1130 |

MeanAbsoluteError: 0.0770971328020096 | Loss: 0.0106580358245022 | Epoch: 1131 |

MeanAbsoluteError: 0.0720335170626640 | Loss: 0.0096212585104028 | Epoch: 1132 |

MeanAbsoluteError: 0.0733764693140984 | Loss: 0.0099265733374826 | Epoch: 1133 |

MeanAbsoluteError: 0.0690442621707916 | Loss: 0.0083087112242720 | Epoch: 1134 |

MeanAbsoluteError: 0.0765947252511978 | Loss: 0.0104372853885191 | Epoch: 1135 |

MeanAbsoluteError: 0.0677077621221542 | Loss: 0.0084910904303736 | Epoch: 1136 |

MeanAbsoluteError: 0.0745299831032753 | Loss: 0.0099079516662217 | Epoch: 1137 |

MeanAbsoluteError: 0.0781344845890999 | Loss: 0.0113178679315509 | Epoch: 1138 |

MeanAbsoluteError: 0.0728972554206848 | Loss: 0.0094153050486299 | Epoch: 1139 |

MeanAbsoluteError: 0.0710478872060776 | Loss: 0.0089030929201787 | Epoch: 1140 |

MeanAbsoluteError: 0.0669848397374153 | Loss: 0.0080258365457363 | Epoch: 1141 |

MeanAbsoluteError: 0.0715565010905266 | Loss: 0.0091548737978640 | Epoch: 1142 |

MeanAbsoluteError: 0.0708602070808411 | Loss: 0.0091273244103650 | Epoch: 1143 |

MeanAbsoluteError: 0.0707661733031273 | Loss: 0.0084934067368279 | Epoch: 1144 |

MeanAbsoluteError: 0.0708365142345428 | Loss: 0.0087682902175978 | Epoch: 1145 |

MeanAbsoluteError: 0.0719301477074623 | Loss: 0.0091117507604865 | Epoch: 1146 |

MeanAbsoluteError: 0.0713092535734177 | Loss: 0.0092337266056469 | Epoch: 1147 |

MeanAbsoluteError: 0.0699829384684563 | Loss: 0.0084455068115494 | Epoch: 1148 |

MeanAbsoluteError: 0.0687723904848099 | Loss: 0.0091803407421685 | Epoch: 1149 |

MeanAbsoluteError: 0.0671131089329720 | Loss: 0.0087500683884415 | Epoch: 1150 |

MeanAbsoluteError: 0.0719459652900696 | Loss: 0.0093504846851526 | Epoch: 1151 |

MeanAbsoluteError: 0.0743340328335762 | Loss: 0.0101055017131209 | Epoch: 1152 |

MeanAbsoluteError: 0.0703700855374336 | Loss: 0.0095029880656754 | Epoch: 1153 |

MeanAbsoluteError: 0.0709020346403122 | Loss: 0.0091121743879315 | Epoch: 1154 |

MeanAbsoluteError: 0.0739380940794945 | Loss: 0.0092778026072968 | Epoch: 1155 |

MeanAbsoluteError: 0.0717305466532707 | Loss: 0.0094344923251386 | Epoch: 1156 |

MeanAbsoluteError: 0.0755668580532074 | Loss: 0.0102404260561525 | Epoch: 1157 |

MeanAbsoluteError: 0.0665167719125748 | Loss: 0.0080044137990141 | Epoch: 1158 |

MeanAbsoluteError: 0.0737614557147026 | Loss: 0.0101033023689524 | Epoch: 1159 |

MeanAbsoluteError: 0.0698042586445808 | Loss: 0.0089558122078112 | Epoch: 1160 |

MeanAbsoluteError: 0.0708379969000816 | Loss: 0.0087991727334762 | Epoch: 1161 |

MeanAbsoluteError: 0.0761933028697968 | Loss: 0.0107606660521318 | Epoch: 1162 |

MeanAbsoluteError: 0.0703091025352478 | Loss: 0.0086571873033730 | Epoch: 1163 |

MeanAbsoluteError: 0.0707952529191971 | Loss: 0.0090408290598134 | Epoch: 1164 |

MeanAbsoluteError: 0.0661467537283897 | Loss: 0.0080320916293158 | Epoch: 1165 |

MeanAbsoluteError: 0.0715991854667664 | Loss: 0.0090736880937038 | Epoch: 1166 |

MeanAbsoluteError: 0.0725976675748825 | Loss: 0.0097558998974879 | Epoch: 1167 |

MeanAbsoluteError: 0.0735879540443420 | Loss: 0.0101218059530947 | Epoch: 1168 |

MeanAbsoluteError: 0.0713030174374580 | Loss: 0.0092109868624296 | Epoch: 1169 |

MeanAbsoluteError: 0.0734420567750931 | Loss: 0.0096720506831965 | Epoch: 1170 |

MeanAbsoluteError: 0.0735661089420319 | Loss: 0.0089929796568898 | Epoch: 1171 |

MeanAbsoluteError: 0.0733410716056824 | Loss: 0.0096806631076015 | Epoch: 1172 |

MeanAbsoluteError: 0.0735263600945473 | Loss: 0.0095509772247109 | Epoch: 1173 |

MeanAbsoluteError: 0.0708796009421349 | Loss: 0.0092483911078792 | Epoch: 1174 |

MeanAbsoluteError: 0.0728058591485023 | Loss: 0.0091239082611143 | Epoch: 1175 |

MeanAbsoluteError: 0.0690005272626877 | Loss: 0.0089129098534553 | Epoch: 1176 |

MeanAbsoluteError: 0.0687796324491501 | Loss: 0.0086335060144484 | Epoch: 1177 |

MeanAbsoluteError: 0.0730583518743515 | Loss: 0.0101112534263666 | Epoch: 1178 |

MeanAbsoluteError: 0.0739194750785828 | Loss: 0.0092704948065026 | Epoch: 1179 |

MeanAbsoluteError: 0.0692448914051056 | Loss: 0.0091311783793450 | Epoch: 1180 |

MeanAbsoluteError: 0.0682249143719673 | Loss: 0.0077948195813709 | Epoch: 1181 |

MeanAbsoluteError: 0.0746102631092072 | Loss: 0.0094276738465608 | Epoch: 1182 |

MeanAbsoluteError: 0.0681728944182396 | Loss: 0.0086403743476452 | Epoch: 1183 |

MeanAbsoluteError: 0.0693699419498444 | Loss: 0.0092196931515840 | Epoch: 1184 |

MeanAbsoluteError: 0.0752856954932213 | Loss: 0.0101704298426436 | Epoch: 1185 |

MeanAbsoluteError: 0.0632870793342590 | Loss: 0.0072724361960475 | Epoch: 1186 |

MeanAbsoluteError: 0.0735668540000916 | Loss: 0.0097458473423709 | Epoch: 1187 |

MeanAbsoluteError: 0.0665733218193054 | Loss: 0.0088264500977190 | Epoch: 1188 |

MeanAbsoluteError: 0.0710608959197998 | Loss: 0.0092791349627320 | Epoch: 1189 |

MeanAbsoluteError: 0.0704220011830330 | Loss: 0.0092359855392109 | Epoch: 1190 |

MeanAbsoluteError: 0.0651642456650734 | Loss: 0.0075645845872547 | Epoch: 1191 |

MeanAbsoluteError: 0.0676573589444160 | Loss: 0.0080162743725911 | Epoch: 1192 |

MeanAbsoluteError: 0.0690322071313858 | Loss: 0.0083130379440809 | Epoch: 1193 |

MeanAbsoluteError: 0.0678318142890930 | Loss: 0.0084162058706958 | Epoch: 1194 |

MeanAbsoluteError: 0.0691720470786095 | Loss: 0.0082022291393271 | Epoch: 1195 |

MeanAbsoluteError: 0.0758125409483910 | Loss: 0.0105129437696451 | Epoch: 1196 |

MeanAbsoluteError: 0.0667120590806007 | Loss: 0.0084848455542427 | Epoch: 1197 |

MeanAbsoluteError: 0.0670898333191872 | Loss: 0.0077767900829713 | Epoch: 1198 |

MeanAbsoluteError: 0.0723942816257477 | Loss: 0.0091439995979817 | Epoch: 1199 |

MeanAbsoluteError: 0.0694923922419548 | Loss: 0.0094884371592555 | Epoch: 1200 |

MeanAbsoluteError: 0.0712788254022598 | Loss: 0.0089867474477796 | Epoch: 1201 |

MeanAbsoluteError: 0.0648735240101814 | Loss: 0.0075425739698464 | Epoch: 1202 |

MeanAbsoluteError: 0.0725867524743080 | Loss: 0.0094600388896652 | Epoch: 1203 |

MeanAbsoluteError: 0.0688964501023293 | Loss: 0.0088374959501622 | Epoch: 1204 |

MeanAbsoluteError: 0.0664794519543648 | Loss: 0.0082911486383333 | Epoch: 1205 |

MeanAbsoluteError: 0.0655982792377472 | Loss: 0.0084157570542069 | Epoch: 1206 |

MeanAbsoluteError: 0.0678544417023659 | Loss: 0.0090636327662893 | Epoch: 1207 |

MeanAbsoluteError: 0.0721559301018715 | Loss: 0.0094533900424722 | Epoch: 1208 |

MeanAbsoluteError: 0.0622016526758671 | Loss: 0.0067169022599167 | Epoch: 1209 |

MeanAbsoluteError: 0.0674180015921593 | Loss: 0.0089908482626439 | Epoch: 1210 |

MeanAbsoluteError: 0.0695351287722588 | Loss: 0.0090114691145754 | Epoch: 1211 |

MeanAbsoluteError: 0.0650373995304108 | Loss: 0.0072134168592068 | Epoch: 1212 |

MeanAbsoluteError: 0.0692954510450363 | Loss: 0.0089630681757990 | Epoch: 1213 |

MeanAbsoluteError: 0.0748051106929779 | Loss: 0.0101131085273907 | Epoch: 1214 |

MeanAbsoluteError: 0.0689355134963989 | Loss: 0.0092268404028679 | Epoch: 1215 |

MeanAbsoluteError: 0.0737222507596016 | Loss: 0.0101098542278730 | Epoch: 1216 |

MeanAbsoluteError: 0.0676701292395592 | Loss: 0.0080728008838681 | Epoch: 1217 |

MeanAbsoluteError: 0.0695484727621078 | Loss: 0.0090794092135426 | Epoch: 1218 |

MeanAbsoluteError: 0.0667334273457527 | Loss: 0.0086289003096075 | Epoch: 1219 |

MeanAbsoluteError: 0.0640681535005569 | Loss: 0.0073190581526190 | Epoch: 1220 |

MeanAbsoluteError: 0.0778120905160904 | Loss: 0.0102683401793183 | Epoch: 1221 |

MeanAbsoluteError: 0.0720247402787209 | Loss: 0.0093078318916863 | Epoch: 1222 |

MeanAbsoluteError: 0.0723437368869781 | Loss: 0.0093220679010604 | Epoch: 1223 |

MeanAbsoluteError: 0.0709807425737381 | Loss: 0.0091675701346442 | Epoch: 1224 |

MeanAbsoluteError: 0.0713775753974915 | Loss: 0.0093865459798205 | Epoch: 1225 |

MeanAbsoluteError: 0.0704527199268341 | Loss: 0.0090108022352676 | Epoch: 1226 |

MeanAbsoluteError: 0.0656383261084557 | Loss: 0.0083695977171252 | Epoch: 1227 |

MeanAbsoluteError: 0.0753611102700233 | Loss: 0.0102924015544704 | Epoch: 1228 |

MeanAbsoluteError: 0.0701126903295517 | Loss: 0.0090481305975118 | Epoch: 1229 |

MeanAbsoluteError: 0.0730990320444107 | Loss: 0.0105546519500270 | Epoch: 1230 |

MeanAbsoluteError: 0.0651449635624886 | Loss: 0.0079959702373071 | Epoch: 1231 |

MeanAbsoluteError: 0.0694925338029861 | Loss: 0.0085180846595601 | Epoch: 1232 |

MeanAbsoluteError: 0.0666825324296951 | Loss: 0.0080074667293350 | Epoch: 1233 |

MeanAbsoluteError: 0.0655047893524170 | Loss: 0.0078616011378714 | Epoch: 1234 |

MeanAbsoluteError: 0.0715290829539299 | Loss: 0.0096715485806029 | Epoch: 1235 |

MeanAbsoluteError: 0.0671703442931175 | Loss: 0.0083176201844860 | Epoch: 1236 |

MeanAbsoluteError: 0.0699932798743248 | Loss: 0.0096389671025948 | Epoch: 1237 |

MeanAbsoluteError: 0.0677242130041122 | Loss: 0.0085804522588114 | Epoch: 1238 |

MeanAbsoluteError: 0.0695762410759926 | Loss: 0.0081709328662691 | Epoch: 1239 |

MeanAbsoluteError: 0.0698123350739479 | Loss: 0.0093136136923567 | Epoch: 1240 |

MeanAbsoluteError: 0.0683453083038330 | Loss: 0.0086917474691775 | Epoch: 1241 |

MeanAbsoluteError: 0.0676895901560783 | Loss: 0.0087146584220955 | Epoch: 1242 |

MeanAbsoluteError: 0.0675857663154602 | Loss: 0.0079030925148133 | Epoch: 1243 |

MeanAbsoluteError: 0.0678542405366898 | Loss: 0.0090135964825216 | Epoch: 1244 |

MeanAbsoluteError: 0.0754652470350266 | Loss: 0.0110325269396011 | Epoch: 1245 |

MeanAbsoluteError: 0.0676740109920502 | Loss: 0.0089280045205184 | Epoch: 1246 |

MeanAbsoluteError: 0.0689785927534103 | Loss: 0.0090947979565802 | Epoch: 1247 |

MeanAbsoluteError: 0.0693435072898865 | Loss: 0.0095896921710452 | Epoch: 1248 |

MeanAbsoluteError: 0.0655711740255356 | Loss: 0.0078810933362668 | Epoch: 1249 |

MeanAbsoluteError: 0.0680463165044785 | Loss: 0.0090941043350055 | Epoch: 1250 |

MeanAbsoluteError: 0.0646833702921867 | Loss: 0.0079773480045462 | Epoch: 1251 |

MeanAbsoluteError: 0.0695341303944588 | Loss: 0.0091768813935050 | Epoch: 1252 |

MeanAbsoluteError: 0.0723897144198418 | Loss: 0.0096052934925804 | Epoch: 1253 |

MeanAbsoluteError: 0.0699800923466682 | Loss: 0.0094651654439561 | Epoch: 1254 |

MeanAbsoluteError: 0.0748149901628494 | Loss: 0.0103388677576368 | Epoch: 1255 |

MeanAbsoluteError: 0.0706802532076836 | Loss: 0.0091762165405089 | Epoch: 1256 |

MeanAbsoluteError: 0.0658210217952728 | Loss: 0.0081147729125557 | Epoch: 1257 |

MeanAbsoluteError: 0.0715734511613846 | Loss: 0.0091425336211493 | Epoch: 1258 |

MeanAbsoluteError: 0.0678317397832870 | Loss: 0.0079324552162628 | Epoch: 1259 |

MeanAbsoluteError: 0.0655079707503319 | Loss: 0.0078485999414746 | Epoch: 1260 |

MeanAbsoluteError: 0.0760006904602051 | Loss: 0.0110741220300467 | Epoch: 1261 |

MeanAbsoluteError: 0.0711369290947914 | Loss: 0.0094518761542516 | Epoch: 1262 |

MeanAbsoluteError: 0.0645487308502197 | Loss: 0.0078292981264409 | Epoch: 1263 |

MeanAbsoluteError: 0.0640544369816780 | Loss: 0.0078533328754823 | Epoch: 1264 |

MeanAbsoluteError: 0.0681712478399277 | Loss: 0.0085240053910790 | Epoch: 1265 |

MeanAbsoluteError: 0.0679940506815910 | Loss: 0.0089414615656521 | Epoch: 1266 |

MeanAbsoluteError: 0.0669510588049889 | Loss: 0.0080082008910298 | Epoch: 1267 |

MeanAbsoluteError: 0.0657232478260994 | Loss: 0.0081272527954692 | Epoch: 1268 |

MeanAbsoluteError: 0.0685509219765663 | Loss: 0.0089792880065700 | Epoch: 1269 |

MeanAbsoluteError: 0.0692617967724800 | Loss: 0.0080941825012754 | Epoch: 1270 |

MeanAbsoluteError: 0.0643127262592316 | Loss: 0.0073227257717129 | Epoch: 1271 |

MeanAbsoluteError: 0.0693370923399925 | Loss: 0.0082023994060485 | Epoch: 1272 |

MeanAbsoluteError: 0.0675821155309677 | Loss: 0.0080976627510487 | Epoch: 1273 |

MeanAbsoluteError: 0.0673690885305405 | Loss: 0.0082584637198640 | Epoch: 1274 |

MeanAbsoluteError: 0.0707979649305344 | Loss: 0.0090332272391728 | Epoch: 1275 |

MeanAbsoluteError: 0.0674105435609818 | Loss: 0.0077424004256439 | Epoch: 1276 |

MeanAbsoluteError: 0.0681989938020706 | Loss: 0.0086218097050369 | Epoch: 1277 |

MeanAbsoluteError: 0.0674866437911987 | Loss: 0.0085005280649542 | Epoch: 1278 |

MeanAbsoluteError: 0.0671639516949654 | Loss: 0.0083256263613854 | Epoch: 1279 |

MeanAbsoluteError: 0.0630234181880951 | Loss: 0.0081991196840439 | Epoch: 1280 |

MeanAbsoluteError: 0.0654474943876266 | Loss: 0.0083551678133275 | Epoch: 1281 |

MeanAbsoluteError: 0.0689337030053139 | Loss: 0.0090341706024628 | Epoch: 1282 |

MeanAbsoluteError: 0.0629124194383621 | Loss: 0.0075137659526930 | Epoch: 1283 |

MeanAbsoluteError: 0.0621831268072128 | Loss: 0.0071266065969879 | Epoch: 1284 |

MeanAbsoluteError: 0.0702346265316010 | Loss: 0.0082153082053325 | Epoch: 1285 |

MeanAbsoluteError: 0.0645866617560387 | Loss: 0.0079498994574533 | Epoch: 1286 |

MeanAbsoluteError: 0.0695939287543297 | Loss: 0.0089412579029158 | Epoch: 1287 |

MeanAbsoluteError: 0.0728226974606514 | Loss: 0.0089424378134572 | Epoch: 1288 |

MeanAbsoluteError: 0.0654545649886131 | Loss: 0.0083984700717580 | Epoch: 1289 |

MeanAbsoluteError: 0.0635727941989899 | Loss: 0.0075470843154471 | Epoch: 1290 |

MeanAbsoluteError: 0.0653821825981140 | Loss: 0.0077729082891892 | Epoch: 1291 |

MeanAbsoluteError: 0.0706762745976448 | Loss: 0.0090435213320112 | Epoch: 1292 |

MeanAbsoluteError: 0.0724355876445770 | Loss: 0.0102611749798719 | Epoch: 1293 |

MeanAbsoluteError: 0.0689788088202477 | Loss: 0.0092645433196837 | Epoch: 1294 |

MeanAbsoluteError: 0.0680117309093475 | Loss: 0.0088657110649486 | Epoch: 1295 |

MeanAbsoluteError: 0.0654366612434387 | Loss: 0.0082979934173879 | Epoch: 1296 |

MeanAbsoluteError: 0.0676603764295578 | Loss: 0.0083546124346564 | Epoch: 1297 |

MeanAbsoluteError: 0.0705934315919876 | Loss: 0.0090451747856908 | Epoch: 1298 |

MeanAbsoluteError: 0.0687427744269371 | Loss: 0.0086981004329088 | Epoch: 1299 |

MeanAbsoluteError: 0.0727434903383255 | Loss: 0.0094710766037557 | Epoch: 1300 |

MeanAbsoluteError: 0.0705679059028625 | Loss: 0.0095606675359159 | Epoch: 1301 |

MeanAbsoluteError: 0.0688144192099571 | Loss: 0.0094463698223020 | Epoch: 1302 |

MeanAbsoluteError: 0.0745014175772667 | Loss: 0.0100277238683217 | Epoch: 1303 |

MeanAbsoluteError: 0.0634368509054184 | Loss: 0.0075667023390436 | Epoch: 1304 |

MeanAbsoluteError: 0.0696567371487617 | Loss: 0.0094187055888021 | Epoch: 1305 |

MeanAbsoluteError: 0.0706652477383614 | Loss: 0.0094941224650393 | Epoch: 1306 |

MeanAbsoluteError: 0.0677551701664925 | Loss: 0.0085527462381530 | Epoch: 1307 |

MeanAbsoluteError: 0.0659917443990707 | Loss: 0.0080453812302828 | Epoch: 1308 |

MeanAbsoluteError: 0.0640876218676567 | Loss: 0.0075252845488042 | Epoch: 1309 |

MeanAbsoluteError: 0.0664003416895866 | Loss: 0.0088671456767770 | Epoch: 1310 |

MeanAbsoluteError: 0.0663144737482071 | Loss: 0.0080531970131657 | Epoch: 1311 |

MeanAbsoluteError: 0.0634544640779495 | Loss: 0.0079215618853535 | Epoch: 1312 |

MeanAbsoluteError: 0.0629883334040642 | Loss: 0.0074207516997315 | Epoch: 1313 |

MeanAbsoluteError: 0.0681820437312126 | Loss: 0.0089186508678540 | Epoch: 1314 |

MeanAbsoluteError: 0.0676098093390465 | Loss: 0.0083668617430521 | Epoch: 1315 |

MeanAbsoluteError: 0.0652496814727783 | Loss: 0.0075829727046463 | Epoch: 1316 |

MeanAbsoluteError: 0.0683088228106499 | Loss: 0.0083458050971967 | Epoch: 1317 |

MeanAbsoluteError: 0.0668292716145515 | Loss: 0.0080924141490565 | Epoch: 1318 |

MeanAbsoluteError: 0.0708575025200844 | Loss: 0.0094735126975744 | Epoch: 1319 |

MeanAbsoluteError: 0.0689029023051262 | Loss: 0.0090884035264025 | Epoch: 1320 |

MeanAbsoluteError: 0.0705532953143120 | Loss: 0.0089705907682461 | Epoch: 1321 |

MeanAbsoluteError: 0.0720054656267166 | Loss: 0.0106686343798113 | Epoch: 1322 |

MeanAbsoluteError: 0.0646906495094299 | Loss: 0.0079822344028920 | Epoch: 1323 |

MeanAbsoluteError: 0.0713153928518295 | Loss: 0.0094255320750623 | Epoch: 1324 |

MeanAbsoluteError: 0.0699148401618004 | Loss: 0.0089810550992358 | Epoch: 1325 |

MeanAbsoluteError: 0.0684277787804604 | Loss: 0.0085602909300845 | Epoch: 1326 |

MeanAbsoluteError: 0.0685327276587486 | Loss: 0.0089783120140419 | Epoch: 1327 |

MeanAbsoluteError: 0.0630204528570175 | Loss: 0.0071405423637286 | Epoch: 1328 |

MeanAbsoluteError: 0.0637871623039246 | Loss: 0.0076257235246764 | Epoch: 1329 |

MeanAbsoluteError: 0.0666264370083809 | Loss: 0.0083140289957676 | Epoch: 1330 |

MeanAbsoluteError: 0.0633303970098495 | Loss: 0.0075683988361016 | Epoch: 1331 |

MeanAbsoluteError: 0.0559964291751385		Loss: 0.0055529360941268		Epoch: 36		MeanAbsoluteError: 0.0559964291751385
MeanAbsoluteError: 0.0512624718248844		Loss: 0.0045191004648682		Epoch: 40		MeanAbsoluteError: 0.0512624718248844
MeanAbsoluteError: 0.0496440082788467		Loss: 0.0048139721421696		Epoch: 44		MeanAbsoluteError: 0.0496440082788467
MeanAbsoluteError: 0.0544668398797512		Loss: 0.0061286149920258		Epoch: 48		MeanAbsoluteError: 0.0544668398797512
MeanAbsoluteError: 0.0571179650723934		Loss: 0.0054987814234521		Epoch: 52		MeanAbsoluteError: 0.0571179650723934
MeanAbsoluteError: 0.0461254231631756		Loss: 0.0037767610996717		Epoch: 56		MeanAbsoluteError: 0.0461254231631756
MeanAbsoluteError: 0.0434462130069733		Loss: 0.0035051909814540		Epoch: 60		MeanAbsoluteError: 0.0434462130069733
MeanAbsoluteError: 0.0497918948531151		Loss: 0.0046373639733678		Epoch: 64		MeanAbsoluteError: 0.0497918948531151
MeanAbsoluteError: 0.0654533877968788		Loss: 0.0067339798705162		Epoch: 68		MeanAbsoluteError: 0.0654533877968788
MeanAbsoluteError: 0.0531934238970280		Loss: 0.0047056918847375		Epoch: 72		MeanAbsoluteError: 0.0531934238970280
MeanAbsoluteError: 0.0452494770288467		Loss: 0.0040486534248645		Epoch: 76		MeanAbsoluteError: 0.0452494770288467
MeanAbsoluteError: 0.0481472462415695		Loss: 0.0039143135807918		Epoch: 80		MeanAbsoluteError: 0.0481472462415695
MeanAbsoluteError: 0.0422858148813248		Loss: 0.0035701094955010		Epoch: 84		MeanAbsoluteError: 0.0422858148813248
MeanAbsoluteError: 0.0440891832113266		Loss: 0.0035781587648671		Epoch: 88		MeanAbsoluteError: 0.0440891832113266
MeanAbsoluteError: 0.0460209809243679		Loss: 0.0041297461721115		Epoch: 92		MeanAbsoluteError: 0.0460209809243679
MeanAbsoluteError: 0.0534663945436478		Loss: 0.0046351083698315		Epoch: 96		MeanAbsoluteError: 0.0534663945436478
MeanAbsoluteError: 0.0557980164885521		Loss: 0.0055767466991184		Epoch: 100		MeanAbsoluteError: 0.0557980164885521
MeanAbsoluteError: 0.0431802272796631		Loss: 0.0035128173936085		Epoch: 104		MeanAbsoluteError: 0.0431802272796631
MeanAbsoluteError: 0.0401490405201912		Loss: 0.0032241009688924		Epoch: 108		MeanAbsoluteError: 0.0401490405201912


```

MeanAbsoluteError: 0.1396099478006363 | Loss: 0.0270088253248679 | Epoch: 15 | MeanAbsoluteError: 0.1396099478006363
MeanAbsoluteError: 0.1524561643600464 | Loss: 0.0292157286680058 | Epoch: 22 | MeanAbsoluteError: 0.1524561643600464
MeanAbsoluteError: 0.0577141307294369 | Loss: 0.0057629092817048 | Epoch: 29 | MeanAbsoluteError: 0.0577141307294369
MeanAbsoluteError: 0.0512868613004684 | Loss: 0.0061870234295432 | Epoch: 36 | MeanAbsoluteError: 0.0512868613004684
MeanAbsoluteError: 0.0898049846291542 | Loss: 0.0114703928610604 | Epoch: 43 | MeanAbsoluteError: 0.0898049846291542
MeanAbsoluteError: 0.0752288550138474 | Loss: 0.0075733111558580 | Epoch: 50 | MeanAbsoluteError: 0.0752288550138474
MeanAbsoluteError: 0.0544765852391720 | Loss: 0.0052043743198738 | Epoch: 57 | MeanAbsoluteError: 0.0544765852391720
MeanAbsoluteError: 0.0928129702806473 | Loss: 0.0113292640859359 | Epoch: 64 | MeanAbsoluteError: 0.0928129702806473
MeanAbsoluteError: 0.0636915341019630 | Loss: 0.0063881949448076 | Epoch: 71 | MeanAbsoluteError: 0.0636915341019630
MeanAbsoluteError: 0.1127697527408600 | Loss: 0.0164719447493553 | Epoch: 78 | MeanAbsoluteError: 0.1127697527408600
MeanAbsoluteError: 0.1295314580202103 | Loss: 0.0201799569063281 | Epoch: 85 | MeanAbsoluteError: 0.1295314580202103
MeanAbsoluteError: 0.0966168344020844 | Loss: 0.0122178531447916 | Epoch: 92 | MeanAbsoluteError: 0.0966168344020844
MeanAbsoluteError: 0.0715004727244377 | Loss: 0.0073310971921800 | Epoch: 99 | MeanAbsoluteError: 0.0715004727244377
MeanAbsoluteError: 0.0751790776848793 | Loss: 0.0084671148981311 | Epoch: 106 | MeanAbsoluteError: 0.0751790776848793
MeanAbsoluteError: 0.0884998217225075 | Loss: 0.0102914346283988 | Epoch: 113 | MeanAbsoluteError: 0.0884998217225075
MeanAbsoluteError: 0.0511729083955288 | Loss: 0.0048638132560116 | Epoch: 120 | MeanAbsoluteError: 0.0511729083955288
Returned to Spot: Validation loss: 0.0036747140356486567

spotPython tuning: 0.0036747140356486567 [#-----] 7.99%

```

```

config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.24181333789384168, 'lr_mult': 1}
Epoch: 1 | MeanAbsoluteError: 0.3089934289455414 | Loss: 0.1247831743798758 | Epoch: 2 | MeanAbsoluteError: 0.3089934289455414

```


MeanAbsoluteError:	0.0514942668378353		Loss:	0.0044579118870101		Epoch:	29		MeanAbsoluteError:	0.0514942668378353
MeanAbsoluteError:	0.0632592514157295		Loss:	0.0063039252264915		Epoch:	31		MeanAbsoluteError:	0.0632592514157295
MeanAbsoluteError:	0.0536349751055241		Loss:	0.0049714456116290		Epoch:	35		MeanAbsoluteError:	0.0536349751055241
MeanAbsoluteError:	0.0557472556829453		Loss:	0.0055166181409732		Epoch:	37		MeanAbsoluteError:	0.0557472556829453
MeanAbsoluteError:	0.0475900284945965		Loss:	0.0045595583260844		Epoch:	41		MeanAbsoluteError:	0.0475900284945965
MeanAbsoluteError:	0.0531024113297462		Loss:	0.0052785903190900		Epoch:	43		MeanAbsoluteError:	0.0531024113297462
MeanAbsoluteError:	0.0456698611378670		Loss:	0.0038869595338934		Epoch:	47		MeanAbsoluteError:	0.0456698611378670
MeanAbsoluteError:	0.0561458840966225		Loss:	0.0053320837231647		Epoch:	49		MeanAbsoluteError:	0.0561458840966225
MeanAbsoluteError:	0.0535153336822987		Loss:	0.0050943880954659		Epoch:	53		MeanAbsoluteError:	0.0535153336822987
MeanAbsoluteError:	0.0444231890141964		Loss:	0.0037923269770353		Epoch:	55		MeanAbsoluteError:	0.0444231890141964
MeanAbsoluteError:	0.0541212223470211		Loss:	0.0051577188393199		Epoch:	59		MeanAbsoluteError:	0.0541212223470211
MeanAbsoluteError:	0.0467403307557106		Loss:	0.0043607883057312		Epoch:	61		MeanAbsoluteError:	0.0467403307557106
MeanAbsoluteError:	0.0463139712810516		Loss:	0.0039849985194834		Epoch:	65		MeanAbsoluteError:	0.0463139712810516
MeanAbsoluteError:	0.0451510064303875		Loss:	0.0037647248858488		Epoch:	67		MeanAbsoluteError:	0.0451510064303875
MeanAbsoluteError:	0.0498130246996880		Loss:	0.0044117303638670		Epoch:	71		MeanAbsoluteError:	0.0498130246996880
MeanAbsoluteError:	0.0416223779320717		Loss:	0.0036121321731786		Epoch:	73		MeanAbsoluteError:	0.0416223779320717
MeanAbsoluteError:	0.0424369461834431		Loss:	0.0038196297031582		Epoch:	77		MeanAbsoluteError:	0.0424369461834431
MeanAbsoluteError:	0.0432984232902527		Loss:	0.0041467740911206		Epoch:	79		MeanAbsoluteError:	0.0432984232902527
MeanAbsoluteError:	0.0454075597226620		Loss:	0.0038750874932463		Epoch:	83		MeanAbsoluteError:	0.0454075597226620


```

MeanAbsoluteError: 0.0584802702069283 | Loss: 0.0055278442971604 | Epoch: 64 | MeanAbsoluteError: 0.0584802702069283
MeanAbsoluteError: 0.0387289598584175 | Loss: 0.0029931662628721 | Epoch: 71 | MeanAbsoluteError: 0.0387289598584175
MeanAbsoluteError: 0.0767731517553329 | Loss: 0.0083690936383056 | Epoch: 78 | MeanAbsoluteError: 0.0767731517553329
MeanAbsoluteError: 0.0666217431426048 | Loss: 0.0071800159976671 | Epoch: 85 | MeanAbsoluteError: 0.0666217431426048
MeanAbsoluteError: 0.0693236812949181 | Loss: 0.0075522858012272 | Epoch: 92 | MeanAbsoluteError: 0.0693236812949181
MeanAbsoluteError: 0.0556882284581661 | Loss: 0.0047371322656737 | Epoch: 99 | MeanAbsoluteError: 0.0556882284581661
MeanAbsoluteError: 0.0661335065960884 | Loss: 0.0072143843250447 | Epoch: 106 | MeanAbsoluteError: 0.0661335065960884
MeanAbsoluteError: 0.0543026030063629 | Loss: 0.0054125258895127 | Epoch: 113 | MeanAbsoluteError: 0.0543026030063629
MeanAbsoluteError: 0.0402885228395462 | Loss: 0.0030285997353004 | Epoch: 120 | MeanAbsoluteError: 0.0402885228395462
MeanAbsoluteError: 0.0482128523290157 | Loss: 0.0036669720368656 | Epoch: 127 | MeanAbsoluteError: 0.0482128523290157
MeanAbsoluteError: 0.0532453283667564 | Loss: 0.0049265027107475 | Epoch: 134 | MeanAbsoluteError: 0.0532453283667564
MeanAbsoluteError: 0.0819845944643021 | Loss: 0.0100394787364884 | Epoch: 141 | MeanAbsoluteError: 0.0819845944643021
Returned to Spot: Validation loss: 0.005200521640577598

spotPython tuning: 0.0036747140356486567 [####-----] 37.66%

config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.25101418251888075, 'lr_mult': 1}
Epoch: 1 | MeanAbsoluteError: 0.1901232004165649 | Loss: 0.0548017342623911 | Epoch: 2 | MeanAbsoluteError: 0.1901232004165649
MeanAbsoluteError: 0.1058278381824493 | Loss: 0.0186099296582765 | Epoch: 8 | MeanAbsoluteError: 0.1058278381824493
MeanAbsoluteError: 0.0953081175684929 | Loss: 0.0156342634980224 | Epoch: 15 | MeanAbsoluteError: 0.0953081175684929
MeanAbsoluteError: 0.0944034457206726 | Loss: 0.0138603430241346 | Epoch: 22 | MeanAbsoluteError: 0.0944034457206726

```



```

MeanAbsoluteError: 0.0779517889022827 | Loss: 0.0114672269280020 | Epoch: 29 | MeanAbsoluteError: 0.0779517889022827
MeanAbsoluteError: 0.1145654171705246 | Loss: 0.0228015212342143 | Epoch: 36 | MeanAbsoluteError: 0.1145654171705246
MeanAbsoluteError: 0.1427936851978302 | Loss: 0.0252846732343498 | Epoch: 43 | MeanAbsoluteError: 0.1427936851978302
MeanAbsoluteError: 0.0776431262493134 | Loss: 0.0093183630685273 | Epoch: 50 | MeanAbsoluteError: 0.0776431262493134
MeanAbsoluteError: 0.1311113387346268 | Loss: 0.0209055725289019 | Epoch: 57 | MeanAbsoluteError: 0.1311113387346268
MeanAbsoluteError: 0.0490914843976498 | Loss: 0.0051877485147040 | Epoch: 64 | MeanAbsoluteError: 0.0490914843976498
MeanAbsoluteError: 0.1065864488482475 | Loss: 0.0141458079022797 | Epoch: 71 | MeanAbsoluteError: 0.1065864488482475
MeanAbsoluteError: 0.0585652813315392 | Loss: 0.0052453716359052 | Epoch: 78 | MeanAbsoluteError: 0.0585652813315392
MeanAbsoluteError: 0.0785874426364899 | Loss: 0.0088096801436653 | Epoch: 85 | MeanAbsoluteError: 0.0785874426364899
MeanAbsoluteError: 0.0528766959905624 | Loss: 0.0045954513388049 | Epoch: 92 | MeanAbsoluteError: 0.0528766959905624
MeanAbsoluteError: 0.0518365018069744 | Loss: 0.0053692959665664 | Epoch: 99 | MeanAbsoluteError: 0.0518365018069744
MeanAbsoluteError: 0.0932385697960854 | Loss: 0.0130063830139606 | Epoch: 106 | MeanAbsoluteError: 0.0932385697960854
Returned to Spot: Validation loss: 0.004095974353779303

spotPython tuning: 0.0036747140356486567 [#####-----] 45.74%

config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.29375246746754513, 'lr_mult': 1}
Epoch: 1 | MeanAbsoluteError: 0.2458864599466324 | Loss: 0.0846163567743803 | Epoch: 2 | MeanAbsoluteError: 0.2458864599466324
MeanAbsoluteError: 0.1149462163448334 | Loss: 0.0214381486079410 | Epoch: 7 | MeanAbsoluteError: 0.1149462163448334
MeanAbsoluteError: 0.0899697244167328 | Loss: 0.0137382013429152 | Epoch: 14 | MeanAbsoluteError: 0.0899697244167328
MeanAbsoluteError: 0.0745254829525948 | Loss: 0.0108294676327588 | Epoch: 21 | MeanAbsoluteError: 0.0745254829525948

```


MeanAbsoluteError:	0.1577263325452805		Loss:	0.0406473375072605		Epoch:	7		MeanAbsoluteError:	0.1577263325452805
MeanAbsoluteError:	0.1379124969244003		Loss:	0.0300107619872219		Epoch:	13		MeanAbsoluteError:	0.1379124969244003
MeanAbsoluteError:	0.1203271895647049		Loss:	0.0224193683580348		Epoch:	19		MeanAbsoluteError:	0.1203271895647049
MeanAbsoluteError:	0.0936883687973022		Loss:	0.0143714835377116		Epoch:	25		MeanAbsoluteError:	0.0936883687973022
MeanAbsoluteError:	0.0964097976684570		Loss:	0.0145498404622470		Epoch:	31		MeanAbsoluteError:	0.0964097976684570
MeanAbsoluteError:	0.0879239365458488		Loss:	0.0134638888320248		Epoch:	37		MeanAbsoluteError:	0.0879239365458488
MeanAbsoluteError:	0.0718226656317711		Loss:	0.0090944591762596		Epoch:	43		MeanAbsoluteError:	0.0718226656317711
MeanAbsoluteError:	0.0738111361861229		Loss:	0.0097957896191235		Epoch:	49		MeanAbsoluteError:	0.0738111361861229
MeanAbsoluteError:	0.0644377246499062		Loss:	0.0064844435056377		Epoch:	55		MeanAbsoluteError:	0.0644377246499062
MeanAbsoluteError:	0.0647642090916634		Loss:	0.0066539663331289		Epoch:	61		MeanAbsoluteError:	0.0647642090916634
MeanAbsoluteError:	0.0580202415585518		Loss:	0.0059056982005897		Epoch:	67		MeanAbsoluteError:	0.0580202415585518
MeanAbsoluteError:	0.0538306571543217		Loss:	0.0046977947993008		Epoch:	73		MeanAbsoluteError:	0.0538306571543217
MeanAbsoluteError:	0.0520225465297699		Loss:	0.0049615671466056		Epoch:	79		MeanAbsoluteError:	0.0520225465297699
MeanAbsoluteError:	0.0474730916321278		Loss:	0.0037846195045859		Epoch:	85		MeanAbsoluteError:	0.0474730916321278
MeanAbsoluteError:	0.0452748425304890		Loss:	0.0037218072929567		Epoch:	91		MeanAbsoluteError:	0.0452748425304890
MeanAbsoluteError:	0.0440431833267212		Loss:	0.0035625744954144		Epoch:	97		MeanAbsoluteError:	0.0440431833267212
MeanAbsoluteError:	0.0415524728596210		Loss:	0.0030889460620911		Epoch:	103		MeanAbsoluteError:	0.0415524728596210
MeanAbsoluteError:	0.0485920757055283		Loss:	0.0042124833102877		Epoch:	109		MeanAbsoluteError:	0.0485920757055283
MeanAbsoluteError:	0.0436604991555214		Loss:	0.0032590332220456		Epoch:	115		MeanAbsoluteError:	0.0436604991555214

MeanAbsoluteError:	0.1292088329792023		Loss:	0.0255617647382774		Epoch:	7		MeanAbsoluteError:
MeanAbsoluteError:	0.1026983931660652		Loss:	0.0160516209312175		Epoch:	13		MeanAbsoluteError:
MeanAbsoluteError:	0.0828437209129333		Loss:	0.0114155325205310		Epoch:	19		MeanAbsoluteError:
MeanAbsoluteError:	0.0846386179327965		Loss:	0.0117996020585691		Epoch:	25		MeanAbsoluteError:
MeanAbsoluteError:	0.0642068460583687		Loss:	0.0073477686260288		Epoch:	31		MeanAbsoluteError:
MeanAbsoluteError:	0.0559007376432419		Loss:	0.0060169003095086		Epoch:	37		MeanAbsoluteError:
MeanAbsoluteError:	0.0509447306394577		Loss:	0.0051212375493426		Epoch:	43		MeanAbsoluteError:
MeanAbsoluteError:	0.0477063879370689		Loss:	0.0044525455634453		Epoch:	49		MeanAbsoluteError:
MeanAbsoluteError:	0.0448210351169109		Loss:	0.0036906276670236		Epoch:	55		MeanAbsoluteError:
MeanAbsoluteError:	0.0441600792109966		Loss:	0.0039660595051062		Epoch:	61		MeanAbsoluteError:
MeanAbsoluteError:	0.0395993217825890		Loss:	0.0032958462834358		Epoch:	67		MeanAbsoluteError:
MeanAbsoluteError:	0.0400504991412163		Loss:	0.0028617895318587		Epoch:	73		MeanAbsoluteError:
MeanAbsoluteError:	0.0338899269700050		Loss:	0.0025086367355758		Epoch:	79		MeanAbsoluteError:
MeanAbsoluteError:	0.0353083163499832		Loss:	0.0026183241769966		Epoch:	85		MeanAbsoluteError:
MeanAbsoluteError:	0.0344848185777664		Loss:	0.0026868901791443		Epoch:	91		MeanAbsoluteError:
MeanAbsoluteError:	0.0344273746013641		Loss:	0.0022956845840733		Epoch:	97		MeanAbsoluteError:
MeanAbsoluteError:	0.0325940474867821		Loss:	0.0019270813637903		Epoch:	103		MeanAbsoluteError:
MeanAbsoluteError:	0.0334756523370743		Loss:	0.0025736705773804		Epoch:	109		MeanAbsoluteError:

MeanAbsoluteError: 0.0368186868727207 | Loss: 0.0032678373972885 | Early stopping at epoch 1
Returned to Spot: Validation loss: 0.003267837397288531

spotPython tuning: 0.0017487917763279064 [#####] 95.49%

config: {'_L_in': 10, '_L_out': 1, 'l1': 128, 'dropout_prob': 0.2634395043987005, 'lr_mult':
Epoch: 1 | MeanAbsoluteError: 0.1046933606266975 | Loss: 0.0181457702756712 | Epoch: 2 | Mean

MeanAbsoluteError: 0.0658888071775436 | Loss: 0.0071162307683967 | Epoch: 6 | MeanAbsoluteEr

MeanAbsoluteError: 0.0774371549487114 | Loss: 0.0088988645002246 | Epoch: 11 | MeanAbsoluteE

MeanAbsoluteError: 0.0698794424533844 | Loss: 0.0074235826828762 | Epoch: 16 | MeanAbsoluteE

MeanAbsoluteError: 0.0518997311592102 | Loss: 0.0045566510433625 | Epoch: 21 | MeanAbsoluteE

MeanAbsoluteError: 0.0555367730557919 | Loss: 0.0047476089829089 | Epoch: 26 | MeanAbsoluteE

MeanAbsoluteError: 0.0407391339540482 | Loss: 0.0027859735082051 | Epoch: 31 | MeanAbsoluteE

MeanAbsoluteError: 0.0408252328634262 | Loss: 0.0030646917937127 | Epoch: 36 | MeanAbsoluteE

MeanAbsoluteError: 0.0424281060695648 | Loss: 0.0027012521719658 | Epoch: 41 | MeanAbsoluteE

MeanAbsoluteError: 0.0380023606121540 | Loss: 0.0023327920046684 | Epoch: 46 | MeanAbsoluteE

MeanAbsoluteError: 0.0383800417184830 | Loss: 0.0022803828622656 | Epoch: 51 | MeanAbsoluteE

MeanAbsoluteError: 0.0400706566870213 | Loss: 0.0026575799688305 | Epoch: 56 | MeanAbsoluteE

MeanAbsoluteError: 0.0542098358273506 | Loss: 0.0039779196101192 | Epoch: 61 | MeanAbsoluteE

MeanAbsoluteError: 0.0495921261608601 | Loss: 0.0034114073067413 | Epoch: 66 | MeanAbsoluteE

MeanAbsoluteError: 0.0277857407927513 | Loss: 0.0013508314916276 | Epoch: 71 | MeanAbsoluteE

MeanAbsoluteError:	0.0287636946886778		Loss:	0.0014772358883515		Epoch:	76		MeanAbsoluteError:	0.0287636946886778
MeanAbsoluteError:	0.0284492969512939		Loss:	0.0013992972151180		Epoch:	81		MeanAbsoluteError:	0.0284492969512939
MeanAbsoluteError:	0.0339997150003910		Loss:	0.0017871692727663		Epoch:	86		MeanAbsoluteError:	0.0339997150003910
MeanAbsoluteError:	0.0254113394767046		Loss:	0.0011533910567921		Epoch:	91		MeanAbsoluteError:	0.0254113394767046
MeanAbsoluteError:	0.0317986533045769		Loss:	0.0016750150274387		Epoch:	96		MeanAbsoluteError:	0.0317986533045769
MeanAbsoluteError:	0.0263713002204895		Loss:	0.0011287978314182		Epoch:	101		MeanAbsoluteError:	0.0263713002204895
MeanAbsoluteError:	0.0248927064239979		Loss:	0.0011340891881111		Epoch:	106		MeanAbsoluteError:	0.0248927064239979
MeanAbsoluteError:	0.0328839570283890		Loss:	0.0016629181949324		Epoch:	111		MeanAbsoluteError:	0.0328839570283890
MeanAbsoluteError:	0.0226477235555649		Loss:	0.0009480327900842		Epoch:	116		MeanAbsoluteError:	0.0226477235555649
MeanAbsoluteError:	0.0345943719148636		Loss:	0.0020424848741018		Epoch:	121		MeanAbsoluteError:	0.0345943719148636
MeanAbsoluteError:	0.0257171001285315		Loss:	0.0011372318641454		Epoch:	126		MeanAbsoluteError:	0.0257171001285315
MeanAbsoluteError:	0.0323606207966805		Loss:	0.0015158604058486		Epoch:	131		MeanAbsoluteError:	0.0323606207966805
MeanAbsoluteError:	0.0257107727229595		Loss:	0.0012141138400981		Epoch:	136		MeanAbsoluteError:	0.0257107727229595
MeanAbsoluteError:	0.0415176153182983		Loss:	0.0025297365776312		Epoch:	141		MeanAbsoluteError:	0.0415176153182983
MeanAbsoluteError:	0.0375518910586834		Loss:	0.0019493569791513		Epoch:	146		MeanAbsoluteError:	0.0375518910586834
MeanAbsoluteError:	0.0237121302634478		Loss:	0.0009161272686661		Epoch:	151		MeanAbsoluteError:	0.0237121302634478
MeanAbsoluteError:	0.0376584716141224		Loss:	0.0020034456637835		Epoch:	156		MeanAbsoluteError:	0.0376584716141224
MeanAbsoluteError:	0.0270745344460011		Loss:	0.0012768083114152		Epoch:	161		MeanAbsoluteError:	0.0270745344460011
MeanAbsoluteError:	0.0297187287360430		Loss:	0.0013305840088594		Epoch:	166		MeanAbsoluteError:	0.0297187287360430

```
MeanAbsoluteError: 0.0264549758285284 | Loss: 0.0012745584045708 | Epoch: 171 | MeanAbsoluteError: 0.0264549758285284
MeanAbsoluteError: 0.0294262543320656 | Loss: 0.0013135057371600 | Epoch: 176 | MeanAbsoluteError: 0.0294262543320656
MeanAbsoluteError: 0.0218083746731281 | Loss: 0.0008090327472401 | Epoch: 181 | MeanAbsoluteError: 0.0218083746731281
MeanAbsoluteError: 0.0304865427315235 | Loss: 0.0014764441446842 | Epoch: 186 | MeanAbsoluteError: 0.0304865427315235
MeanAbsoluteError: 0.0219854563474655 | Loss: 0.0009052643083698 | Epoch: 191 | MeanAbsoluteError: 0.0219854563474655
MeanAbsoluteError: 0.0265823714435101 | Loss: 0.0012094502790684 | Epoch: 196 | MeanAbsoluteError: 0.0265823714435101
MeanAbsoluteError: 0.0369167216122150 | Loss: 0.0018721219630128 | Epoch: 201 | MeanAbsoluteError: 0.0369167216122150
MeanAbsoluteError: 0.0231902096420527 | Loss: 0.0008655194113472 | Epoch: 206 | MeanAbsoluteError: 0.0231902096420527
MeanAbsoluteError: 0.0267195422202349 | Loss: 0.0010858270627643 | Epoch: 211 | MeanAbsoluteError: 0.0267195422202349
MeanAbsoluteError: 0.0225965380668640 | Loss: 0.0009895818932962 | Epoch: 216 | MeanAbsoluteError: 0.0225965380668640
MeanAbsoluteError: 0.0318903774023056 | Loss: 0.0015177360558147 | Epoch: 221 | MeanAbsoluteError: 0.0318903774023056
MeanAbsoluteError: 0.0245253778994083 | Loss: 0.0010248127785560 | Epoch: 226 | MeanAbsoluteError: 0.0245253778994083
MeanAbsoluteError: 0.0271553806960583 | Loss: 0.0012434169735858 | Epoch: 231 | MeanAbsoluteError: 0.0271553806960583
MeanAbsoluteError: 0.0211309380829334 | Loss: 0.0008726530037453 | Epoch: 236 | MeanAbsoluteError: 0.0211309380829334
MeanAbsoluteError: 0.0248939059674740 | Loss: 0.0010125685979514 | Epoch: 241 | MeanAbsoluteError: 0.0248939059674740
MeanAbsoluteError: 0.0278821755200624 | Loss: 0.0013848338597822 | Epoch: 246 | MeanAbsoluteError: 0.0278821755200624
MeanAbsoluteError: 0.0409895181655884 | Loss: 0.0023926682770252 | Epoch: 251 | MeanAbsoluteError: 0.0409895181655884
MeanAbsoluteError: 0.0266578327864408 | Loss: 0.0012575217651350 | Epoch: 256 | MeanAbsoluteError: 0.0266578327864408
MeanAbsoluteError: 0.0259793307632208 | Loss: 0.0011089971889497 | Epoch: 261 | MeanAbsoluteError: 0.0259793307632208
Returned to Spot: Validation loss: 0.0008745023162765918

spotPython tuning: 0.0008745023162765918 [#####] 100.00% Done...

<spotPython.spot.spot.Spot at 0x185f6f1c0>
```

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")
```

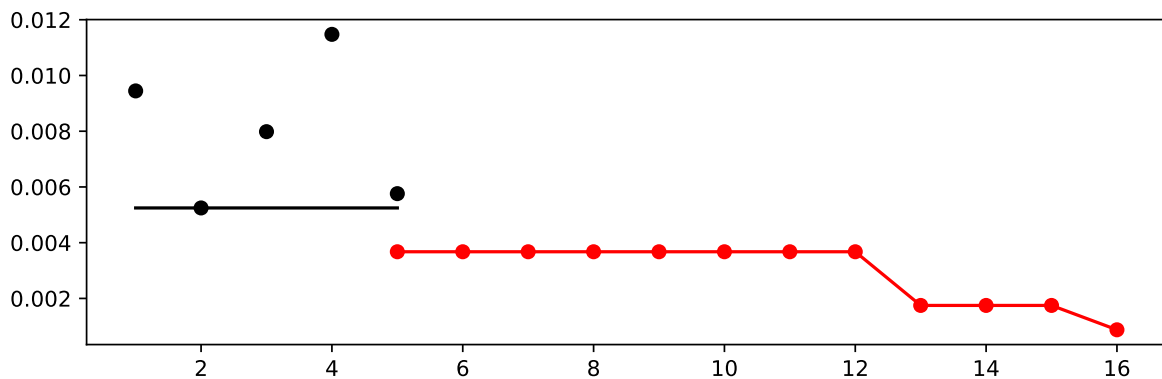


Figure 19.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

```
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	7.0	transform_po
dropout_prob	float	0.01	0.0	0.9	0.2634395043987005	None
lr_mult	float	1.0	0.1	10.0	3.9421338438887745	None
batch_size	int	4	1.0	4.0	4.0	transform_po
epochs	int	4	2.0	16.0	11.0	transform_po

k_folds	int	1		1.0		1.0		1.0	None
patience	int	2		3.0		7.0		6.0	transform_po
optimizer	factor	SGD		0.0		6.0		3.0	None
sgd_momentum	float	0.0		0.0		1.0		0.23165814493788056	None

```
spot_tuner.plot_importance(threshold=0.025,
                           filename="./figures/" + experiment_name+"_importance.png")
```

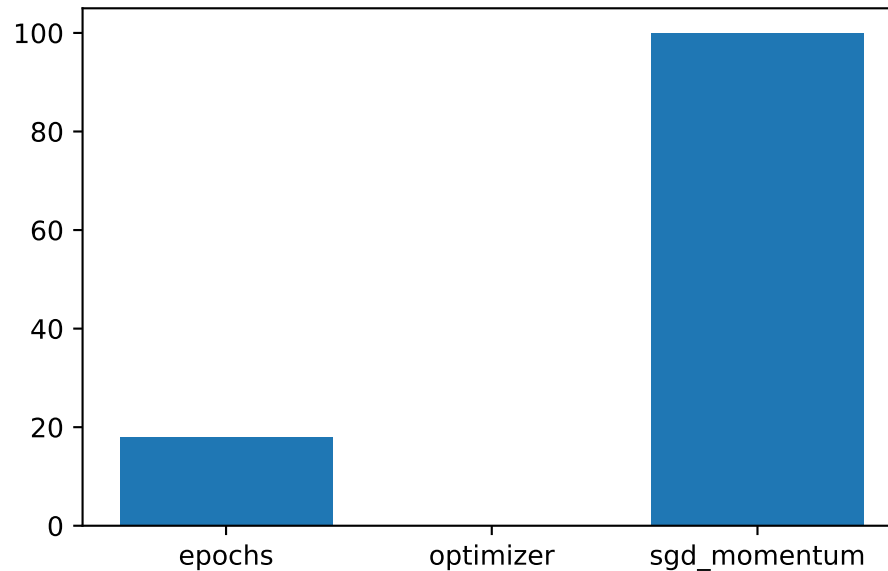


Figure 19.2: Variable importance plot, threshold 0.025.

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=128, bias=True)
  (fc2): Linear(in_features=128, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=1, bias=True)
  (relu): ReLU()
```

```

    (softmax): Softmax(dim=1)
    (dropout1): Dropout(p=0.2634395043987005, inplace=False)
    (dropout2): Dropout(p=0.13171975219935025, 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: 0.1252194792032242 | Loss: 0.0243228561686058 | Epoch: 2 | Mean
MeanAbsoluteError: 0.0721857845783234 | Loss: 0.0082443502829655 | Epoch: 6 | MeanAbsoluteEr
MeanAbsoluteError: 0.0714354515075684 | Loss: 0.0079417246835012 | Epoch: 11 | MeanAbsoluteE
MeanAbsoluteError: 0.0560701489448547 | Loss: 0.0050193272825134 | Epoch: 16 | MeanAbsoluteE
MeanAbsoluteError: 0.0479538254439831 | Loss: 0.0041032580928387 | Epoch: 21 | MeanAbsoluteE
MeanAbsoluteError: 0.0491281822323799 | Loss: 0.0039316917598051 | Epoch: 26 | MeanAbsoluteE
MeanAbsoluteError: 0.0507984980940819 | Loss: 0.0040671861191329 | Epoch: 31 | MeanAbsoluteE
MeanAbsoluteError: 0.0421102456748486 | Loss: 0.0029855988748176 | Epoch: 36 | MeanAbsoluteE
MeanAbsoluteError: 0.0380061231553555 | Loss: 0.0022671862787224 | Epoch: 41 | MeanAbsoluteE

```

MeanAbsoluteError:	0.0384438782930374		Loss:	0.0024717473271793		Epoch:	46		MeanAbsoluteError:	0.0384438782930374
MeanAbsoluteError:	0.0359942354261875		Loss:	0.0023448975499425		Epoch:	51		MeanAbsoluteError:	0.0359942354261875
MeanAbsoluteError:	0.0357487350702286		Loss:	0.0020348153346659		Epoch:	56		MeanAbsoluteError:	0.0357487350702286
MeanAbsoluteError:	0.0345531739294529		Loss:	0.0020773877990187		Epoch:	61		MeanAbsoluteError:	0.0345531739294529
MeanAbsoluteError:	0.0336252376437187		Loss:	0.0018568480675305		Epoch:	66		MeanAbsoluteError:	0.0336252376437187
MeanAbsoluteError:	0.0309494491666555		Loss:	0.0015037746113529		Epoch:	71		MeanAbsoluteError:	0.0309494491666555
MeanAbsoluteError:	0.0294891390949488		Loss:	0.0015833789366297		Epoch:	76		MeanAbsoluteError:	0.0294891390949488
MeanAbsoluteError:	0.0301689784973860		Loss:	0.0016190598482928		Epoch:	81		MeanAbsoluteError:	0.0301689784973860
MeanAbsoluteError:	0.0293881259858608		Loss:	0.0015315828832651		Epoch:	86		MeanAbsoluteError:	0.0293881259858608
MeanAbsoluteError:	0.0274575073271990		Loss:	0.0011767613683141		Epoch:	91		MeanAbsoluteError:	0.0274575073271990
MeanAbsoluteError:	0.0263920109719038		Loss:	0.0012761728767624		Epoch:	96		MeanAbsoluteError:	0.0263920109719038
MeanAbsoluteError:	0.0346009507775307		Loss:	0.0020784954387253		Epoch:	101		MeanAbsoluteError:	0.0346009507775307
MeanAbsoluteError:	0.0273224320262671		Loss:	0.0013638882312654		Epoch:	106		MeanAbsoluteError:	0.0273224320262671
MeanAbsoluteError:	0.0279570482671261		Loss:	0.0013383323890402		Epoch:	111		MeanAbsoluteError:	0.0279570482671261
MeanAbsoluteError:	0.0376980006694794		Loss:	0.0022406751093896		Epoch:	116		MeanAbsoluteError:	0.0376980006694794
MeanAbsoluteError:	0.0358993820846081		Loss:	0.0019214566380374		Epoch:	121		MeanAbsoluteError:	0.0358993820846081
MeanAbsoluteError:	0.0237220618873835		Loss:	0.0009346610428398		Epoch:	126		MeanAbsoluteError:	0.0237220618873835
MeanAbsoluteError:	0.0270416252315044		Loss:	0.0013434576134099		Epoch:	131		MeanAbsoluteError:	0.0270416252315044
MeanAbsoluteError:	0.0228122696280479		Loss:	0.0009541582411131		Epoch:	136		MeanAbsoluteError:	0.0228122696280479

MeanAbsoluteError:	0.0254673585295677		Loss:	0.0014124156977663		Epoch:	141		MeanAbsoluteError:
MeanAbsoluteError:	0.0253842175006866		Loss:	0.0011305071116352		Epoch:	146		MeanAbsoluteError:
MeanAbsoluteError:	0.0291923563927412		Loss:	0.0013663995839459		Epoch:	151		MeanAbsoluteError:
MeanAbsoluteError:	0.0282467585057020		Loss:	0.0012947748090435		Epoch:	156		MeanAbsoluteError:
MeanAbsoluteError:	0.0260330494493246		Loss:	0.0014194816543941		Epoch:	161		MeanAbsoluteError:
MeanAbsoluteError:	0.0280468631535769		Loss:	0.0012690545485576		Epoch:	166		MeanAbsoluteError:
MeanAbsoluteError:	0.0236350856721401		Loss:	0.0010209698858058		Epoch:	171		MeanAbsoluteError:
MeanAbsoluteError:	0.0335752964019775		Loss:	0.0017888179588083		Epoch:	176		MeanAbsoluteError:
MeanAbsoluteError:	0.0235887691378593		Loss:	0.0010730549233573		Epoch:	181		MeanAbsoluteError:
MeanAbsoluteError:	0.0264898315072060		Loss:	0.0012538753370264		Epoch:	186		MeanAbsoluteError:
MeanAbsoluteError:	0.0232206173241138		Loss:	0.0009596806883469		Epoch:	191		MeanAbsoluteError:
MeanAbsoluteError:	0.0221656169742346		Loss:	0.0008931523648483		Epoch:	196		MeanAbsoluteError:
MeanAbsoluteError:	0.0414316132664680		Loss:	0.0027793893639586		Epoch:	201		MeanAbsoluteError:
MeanAbsoluteError:	0.0306473486125469		Loss:	0.0017648755732041		Epoch:	206		MeanAbsoluteError:
MeanAbsoluteError:	0.0416488833725452		Loss:	0.0026320483547782		Epoch:	211		MeanAbsoluteError:
MeanAbsoluteError:	0.0298693664371967		Loss:	0.0014684361613993		Epoch:	216		MeanAbsoluteError:
MeanAbsoluteError:	0.0288183633238077		Loss:	0.0014099736673463		Epoch:	221		MeanAbsoluteError:
MeanAbsoluteError:	0.0366386584937572		Loss:	0.0021525712616399		Epoch:	226		MeanAbsoluteError:
MeanAbsoluteError:	0.0366585999727249		Loss:	0.0020152368386717		Epoch:	231		MeanAbsoluteError:

MeanAbsoluteError:	0.0260509066283703		Loss:	0.0011709413527952		Epoch:	236		MeanAbsoluteError:
MeanAbsoluteError:	0.0239223763346672		Loss:	0.0010277660805609		Epoch:	241		MeanAbsoluteError:
MeanAbsoluteError:	0.0361624807119370		Loss:	0.0018712167297245		Epoch:	246		MeanAbsoluteError:
MeanAbsoluteError:	0.0446558073163033		Loss:	0.0026259524686458		Epoch:	251		MeanAbsoluteError:
MeanAbsoluteError:	0.0211274120956659		Loss:	0.0008072376260411		Epoch:	256		MeanAbsoluteError:
MeanAbsoluteError:	0.0266616977751255		Loss:	0.0013912559799409		Epoch:	261		MeanAbsoluteError:
MeanAbsoluteError:	0.0272515453398228		Loss:	0.0012772698659989		Epoch:	266		MeanAbsoluteError:
MeanAbsoluteError:	0.0243388824164867		Loss:	0.0011038206621857		Epoch:	271		MeanAbsoluteError:
MeanAbsoluteError:	0.0229431279003620		Loss:	0.0009900194418151		Epoch:	276		MeanAbsoluteError:
MeanAbsoluteError:	0.0244472697377205		Loss:	0.0010086593545949		Epoch:	281		MeanAbsoluteError:
MeanAbsoluteError:	0.0243386942893267		Loss:	0.0011018056722701		Epoch:	286		MeanAbsoluteError:
MeanAbsoluteError:	0.0243123080581427		Loss:	0.0011344176975973		Epoch:	291		MeanAbsoluteError:
MeanAbsoluteError:	0.0207091923803091		Loss:	0.0007270965893679		Epoch:	296		MeanAbsoluteError:
MeanAbsoluteError:	0.0212738942354918		Loss:	0.0007927403524886		Epoch:	301		MeanAbsoluteError:
MeanAbsoluteError:	0.0329827256500721		Loss:	0.0016948360686288		Epoch:	306		MeanAbsoluteError:
MeanAbsoluteError:	0.0263507775962353		Loss:	0.0014104927142494		Epoch:	311		MeanAbsoluteError:
MeanAbsoluteError:	0.0239754691720009		Loss:	0.0010707563411224		Epoch:	316		MeanAbsoluteError:
MeanAbsoluteError:	0.0244414452463388		Loss:	0.0011520307560108		Epoch:	321		MeanAbsoluteError:
MeanAbsoluteError:	0.0406668074429035		Loss:	0.0022979741364619		Epoch:	326		MeanAbsoluteError:


```

MeanAbsoluteError: 0.0257289186120033 | Loss: 0.0011142801545487 | Epoch: 331 | MeanAbsoluteError: 0.0257289186120033
MeanAbsoluteError: 0.0266763903200626 | Loss: 0.0012106926174295 | Epoch: 336 | MeanAbsoluteError: 0.0266763903200626
MeanAbsoluteError: 0.0326016917824745 | Loss: 0.0018571127791227 | Epoch: 341 | MeanAbsoluteError: 0.0326016917824745
MeanAbsoluteError: 0.0280104316771030 | Loss: 0.0012809108072696 | Epoch: 346 | MeanAbsoluteError: 0.0280104316771030
MeanAbsoluteError: 0.0231843721121550 | Loss: 0.0009573515348357 | Epoch: 351 | MeanAbsoluteError: 0.0231843721121550
MeanAbsoluteError: 0.0264734402298927 | Loss: 0.0012484832427857 | Epoch: 356 | MeanAbsoluteError: 0.0264734402298927
Returned to Spot: Validation loss: 0.0018957655227399971

```

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: 0.0394475348293781 | Loss: 0.0022895976580912 | Final evaluation: Validation loss: 0.0022895976580912
Final evaluation: Validation metric: 0.03944753482937813
-----

```

```

(0.0022895976580912247, nan, tensor(0.0394))

```

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: 0.1053961589932442 | Loss: 0.0197596665737884 | Epoch: 2 | Mean
MeanAbsoluteError: 0.0984850674867630 | Loss: 0.0130856352126492 | Epoch: 4 | MeanAbsoluteEr
MeanAbsoluteError: 0.0646924301981926 | Loss: 0.0072300777371441 | Epoch: 7 | MeanAbsoluteEr
MeanAbsoluteError: 0.0739994868636131 | Loss: 0.0078997024601059 | Epoch: 10 | MeanAbsoluteE
MeanAbsoluteError: 0.0493783317506313 | Loss: 0.0043399363223995 | Epoch: 13 | MeanAbsoluteE
MeanAbsoluteError: 0.0557980053126812 | Loss: 0.0048108836469640 | Epoch: 16 | MeanAbsoluteE
MeanAbsoluteError: 0.0438386201858521 | Loss: 0.0036275155725889 | Epoch: 19 | MeanAbsoluteE
MeanAbsoluteError: 0.0408801883459091 | Loss: 0.0031612551704581 | Epoch: 22 | MeanAbsoluteE
MeanAbsoluteError: 0.0486405789852142 | Loss: 0.0036274595518730 | Epoch: 25 | MeanAbsoluteE
MeanAbsoluteError: 0.0394966304302216 | Loss: 0.0024355933003660 | Epoch: 28 | MeanAbsoluteE
MeanAbsoluteError: 0.0380466096103191 | Loss: 0.0030415309759389 | Epoch: 31 | MeanAbsoluteE
MeanAbsoluteError: 0.0290259066969156 | Loss: 0.0015826289475496 | Epoch: 34 | MeanAbsoluteE
MeanAbsoluteError: 0.0352232381701469 | Loss: 0.0022848651611379 | Epoch: 37 | MeanAbsoluteE
MeanAbsoluteError: 0.0311474967747927 | Loss: 0.0019274761517798 | Epoch: 40 | MeanAbsoluteE
MeanAbsoluteError: 0.0329090729355812 | Loss: 0.0020777402179582 | Epoch: 43 | MeanAbsoluteE

```

MeanAbsoluteError:	0.0300631970167160		Loss:	0.0025639821209812		Epoch:	46		MeanAbsoluteError:	0.0280145816504955
MeanAbsoluteError:	0.0280145816504955		Loss:	0.0012657527279641		Epoch:	49		MeanAbsoluteError:	0.0393049232661724
MeanAbsoluteError:	0.0393049232661724		Loss:	0.0023491384095645		Epoch:	52		MeanAbsoluteError:	0.0271846707910299
MeanAbsoluteError:	0.0271846707910299		Loss:	0.0011557851851519		Epoch:	55		MeanAbsoluteError:	0.0258029364049435
MeanAbsoluteError:	0.0258029364049435		Loss:	0.0010743194475903		Epoch:	58		MeanAbsoluteError:	0.0273527167737484
MeanAbsoluteError:	0.0273527167737484		Loss:	0.0012906211611283		Epoch:	61		MeanAbsoluteError:	0.0262058302760124
MeanAbsoluteError:	0.0262058302760124		Loss:	0.0011308465078140		Epoch:	64		MeanAbsoluteError:	0.0258859321475029
MeanAbsoluteError:	0.0258859321475029		Loss:	0.0011267005549079		Epoch:	67		MeanAbsoluteError:	0.0261350534856319
MeanAbsoluteError:	0.0261350534856319		Loss:	0.0010325299226679		Epoch:	70		MeanAbsoluteError:	0.0287544559687376
MeanAbsoluteError:	0.0287544559687376		Loss:	0.0013508352234827		Epoch:	73		MeanAbsoluteError:	0.0283468365669250
MeanAbsoluteError:	0.0283468365669250		Loss:	0.0013093185261823		Epoch:	76		MeanAbsoluteError:	0.0234428364783525
MeanAbsoluteError:	0.0234428364783525		Loss:	0.0007148072685855		Epoch:	79		MeanAbsoluteError:	0.0309417843818665
MeanAbsoluteError:	0.0309417843818665		Loss:	0.0017154401118335		Epoch:	82		MeanAbsoluteError:	0.0320538729429245
MeanAbsoluteError:	0.0320538729429245		Loss:	0.0015879282977299		Epoch:	85		MeanAbsoluteError:	0.0272529311478138
MeanAbsoluteError:	0.0272529311478138		Loss:	0.0018053537184772		Epoch:	88		MeanAbsoluteError:	0.0221117362380028
MeanAbsoluteError:	0.0221117362380028		Loss:	0.0008241557349850		Epoch:	91		MeanAbsoluteError:	0.0275430791079998
MeanAbsoluteError:	0.0275430791079998		Loss:	0.0011734503454396		Epoch:	94		MeanAbsoluteError:	0.0312458164989948
MeanAbsoluteError:	0.0312458164989948		Loss:	0.0016421444597654		Epoch:	97		MeanAbsoluteError:	0.0252871587872505
MeanAbsoluteError:	0.0252871587872505		Loss:	0.0011078122188337		Epoch:	100		MeanAbsoluteError:	

MeanAbsoluteError: 0.0457737334072590		Loss: 0.0027709152948643		Epoch: 103		MeanAbsoluteError: 0.0264397356659174
MeanAbsoluteError: 0.0264397356659174		Loss: 0.0011174482975288		Epoch: 106		MeanAbsoluteError: 0.0296921283006668
MeanAbsoluteError: 0.0296921283006668		Loss: 0.0012483263729207		Epoch: 109		MeanAbsoluteError: 0.0267754867672920
MeanAbsoluteError: 0.0267754867672920		Loss: 0.0013565002840811		Epoch: 112		MeanAbsoluteError: 0.0271700639277697
MeanAbsoluteError: 0.0271700639277697		Loss: 0.0015319323283620		Epoch: 115		MeanAbsoluteError: 0.0275086499750614
MeanAbsoluteError: 0.0275086499750614		Loss: 0.0012293787357131		Epoch: 118		MeanAbsoluteError: 0.0291009806096554
MeanAbsoluteError: 0.0291009806096554		Loss: 0.0014370331025150		Epoch: 121		MeanAbsoluteError: 0.0258851889520884
MeanAbsoluteError: 0.0258851889520884		Loss: 0.0013574452189329		Epoch: 124		MeanAbsoluteError: 0.0225768331438303
MeanAbsoluteError: 0.0225768331438303		Loss: 0.0007929598130951		Epoch: 127		MeanAbsoluteError: 0.0309092830866575
MeanAbsoluteError: 0.0309092830866575		Loss: 0.0013568848537813		Epoch: 130		MeanAbsoluteError: 0.0230264328420162
MeanAbsoluteError: 0.0230264328420162		Loss: 0.0008218187057147		Epoch: 133		MeanAbsoluteError: 0.0267968364059925
MeanAbsoluteError: 0.0267968364059925		Loss: 0.0012040701944248		Epoch: 136		MeanAbsoluteError: 0.0244129896163940
MeanAbsoluteError: 0.0244129896163940		Loss: 0.0014973777080221		Epoch: 139		MeanAbsoluteError: 0.0326274633407593
MeanAbsoluteError: 0.0326274633407593		Loss: 0.0016979707538017		Epoch: 142		MeanAbsoluteError: 0.0296807345002890
MeanAbsoluteError: 0.0296807345002890		Loss: 0.0016330574830395		Epoch: 145		MeanAbsoluteError: 0.0241817068308592
MeanAbsoluteError: 0.0241817068308592		Loss: 0.0011557241793655		Epoch: 148		MeanAbsoluteError: 0.0194849427789450
MeanAbsoluteError: 0.0194849427789450		Loss: 0.0008455674937328		Epoch: 151		MeanAbsoluteError: 0.0252529047429562
MeanAbsoluteError: 0.0252529047429562		Loss: 0.0009776151502073		Epoch: 154		MeanAbsoluteError: 0.0245256535708904
MeanAbsoluteError: 0.0245256535708904		Loss: 0.0010402316194294		Epoch: 157		

MeanAbsoluteError:	0.0265404488891363	Loss:	0.0011830042016559	Epoch:	160	MeanAbsoluteE
MeanAbsoluteError:	0.0258140731602907	Loss:	0.0012261834719019	Epoch:	163	MeanAbsoluteE
MeanAbsoluteError:	0.0243560895323753	Loss:	0.0009588603237976	Epoch:	166	MeanAbsoluteE
MeanAbsoluteError:	0.0263361241668463	Loss:	0.0012456858613794	Epoch:	169	MeanAbsoluteE
MeanAbsoluteError:	0.0259733721613884	Loss:	0.0011950703420942	Epoch:	172	MeanAbsoluteE
MeanAbsoluteError:	0.0341915041208267	Loss:	0.0017195255828223	Epoch:	175	MeanAbsoluteE
MeanAbsoluteError:	0.0237789135426283	Loss:	0.0008548844614000	Epoch:	178	MeanAbsoluteE
MeanAbsoluteError:	0.0367132313549519	Loss:	0.0018288491327049	Epoch:	181	MeanAbsoluteE
MeanAbsoluteError:	0.0308192372322083	Loss:	0.0016011052648537	Epoch:	184	MeanAbsoluteE
MeanAbsoluteError:	0.0233438722789288	Loss:	0.0008061706487622	Epoch:	187	MeanAbsoluteE
Fold: 2						
Epoch: 1						
MeanAbsoluteError:	0.1148984432220459	Loss:	0.0220737539763962	Epoch:	2	MeanAbsoluteEr
MeanAbsoluteError:	0.0912838950753212	Loss:	0.0144355476700834	Epoch:	5	MeanAbsoluteErr
MeanAbsoluteError:	0.0693705677986145	Loss:	0.0083008607782956	Epoch:	8	MeanAbsoluteErr
MeanAbsoluteError:	0.0454918146133423	Loss:	0.0035595260560513	Epoch:	11	MeanAbsoluteEs
MeanAbsoluteError:	0.0451785176992416	Loss:	0.0032362374477088	Epoch:	14	MeanAbsoluteE
MeanAbsoluteError:	0.0660772100090981	Loss:	0.0058004187553057	Epoch:	17	MeanAbsoluteE
MeanAbsoluteError:	0.0369143709540367	Loss:	0.0024590443021485	Epoch:	20	MeanAbsoluteE
MeanAbsoluteError:	0.0337305516004562	Loss:	0.0018153334468869	Epoch:	23	MeanAbsoluteE

MeanAbsoluteError:	0.0327437743544579		Loss:	0.0017223255791967		Epoch:	26		MeanAbsoluteError:	0.0327437743544579
MeanAbsoluteError:	0.0333147309720516		Loss:	0.0021771199119809		Epoch:	29		MeanAbsoluteError:	0.0333147309720516
MeanAbsoluteError:	0.0317563377320766		Loss:	0.0018053214963792		Epoch:	32		MeanAbsoluteError:	0.0317563377320766
MeanAbsoluteError:	0.0395802967250347		Loss:	0.0029609768618164		Epoch:	35		MeanAbsoluteError:	0.0395802967250347
MeanAbsoluteError:	0.0323693118989468		Loss:	0.0019984408614359		Epoch:	38		MeanAbsoluteError:	0.0323693118989468
MeanAbsoluteError:	0.0475719012320042		Loss:	0.0036490621444370		Epoch:	41		MeanAbsoluteError:	0.0475719012320042
MeanAbsoluteError:	0.0292224548757076		Loss:	0.0016105542308651		Epoch:	44		MeanAbsoluteError:	0.0292224548757076
MeanAbsoluteError:	0.0299448203295469		Loss:	0.0018073161398726		Epoch:	47		MeanAbsoluteError:	0.0299448203295469
MeanAbsoluteError:	0.0279204510152340		Loss:	0.0014549218633744		Epoch:	50		MeanAbsoluteError:	0.0279204510152340
MeanAbsoluteError:	0.0266552250832319		Loss:	0.0015486988455190		Epoch:	53		MeanAbsoluteError:	0.0266552250832319
MeanAbsoluteError:	0.0319711081683636		Loss:	0.0016804191816066		Epoch:	56		MeanAbsoluteError:	0.0319711081683636
MeanAbsoluteError:	0.0333864949643612		Loss:	0.0017864696107738		Epoch:	59		MeanAbsoluteError:	0.0333864949643612
MeanAbsoluteError:	0.0393760688602924		Loss:	0.0029724697482639		Epoch:	62		MeanAbsoluteError:	0.0393760688602924
MeanAbsoluteError:	0.0295269675552845		Loss:	0.0015862466285138		Epoch:	65		MeanAbsoluteError:	0.0295269675552845
MeanAbsoluteError:	0.0277321152389050		Loss:	0.0016305324970745		Epoch:	68		MeanAbsoluteError:	0.0277321152389050
MeanAbsoluteError:	0.0283204168081284		Loss:	0.0014134764954049		Epoch:	71		MeanAbsoluteError:	0.0283204168081284
MeanAbsoluteError:	0.0273362137377262		Loss:	0.0011467103280925		Epoch:	74		MeanAbsoluteError:	0.0273362137377262
MeanAbsoluteError:	0.0269251056015491		Loss:	0.0015214507750768		Epoch:	77		MeanAbsoluteError:	0.0269251056015491
MeanAbsoluteError:	0.0527560003101826		Loss:	0.0034887758915180		Epoch:	80		MeanAbsoluteError:	0.0527560003101826

MeanAbsoluteError:	0.0301510374993086		Loss:	0.0018124628945121		Epoch:	83		MeanAbsoluteError:
MeanAbsoluteError:	0.0304139144718647		Loss:	0.0018021411545176		Epoch:	86		MeanAbsoluteError:
MeanAbsoluteError:	0.0353985205292702		Loss:	0.0019629326083564		Epoch:	89		MeanAbsoluteError:
MeanAbsoluteError:	0.0286320298910141		Loss:	0.0012456011817059		Epoch:	92		MeanAbsoluteError:
MeanAbsoluteError:	0.0318809263408184		Loss:	0.0016359989281877		Epoch:	95		MeanAbsoluteError:
MeanAbsoluteError:	0.0331305190920830		Loss:	0.0021757120931787		Epoch:	98		MeanAbsoluteError:
MeanAbsoluteError:	0.0274778176099062		Loss:	0.0014910442904303		Epoch:	101		MeanAbsoluteError:
MeanAbsoluteError:	0.0252929981797934		Loss:	0.0011851265860189		Epoch:	104		MeanAbsoluteError:
MeanAbsoluteError:	0.0275972317904234		Loss:	0.0028356793627609		Epoch:	107		MeanAbsoluteError:
MeanAbsoluteError:	0.0310302749276161		Loss:	0.0014958784990345		Epoch:	110		MeanAbsoluteError:
MeanAbsoluteError:	0.0285016186535358		Loss:	0.0014187872501290		Epoch:	113		MeanAbsoluteError:
MeanAbsoluteError:	0.0296462830156088		Loss:	0.0016023798130586		Epoch:	116		MeanAbsoluteError:
MeanAbsoluteError:	0.0272224992513657		Loss:	0.0013120392104611		Epoch:	119		MeanAbsoluteError:
MeanAbsoluteError:	0.0306347366422415		Loss:	0.0017899122623411		Epoch:	122		MeanAbsoluteError:
MeanAbsoluteError:	0.0286495443433523		Loss:	0.0013948816922493		Epoch:	125		MeanAbsoluteError:
MeanAbsoluteError:	0.0337501913309097		Loss:	0.0020409593957343		Epoch:	128		MeanAbsoluteError:
MeanAbsoluteError:	0.0291549209505320		Loss:	0.0018635429026160		Epoch:	131		MeanAbsoluteError:
MeanAbsoluteError:	0.0305640697479248		Loss:	0.0018298101744481		Epoch:	134		MeanAbsoluteError:
MeanAbsoluteError:	0.0315228216350079		Loss:	0.0017358031847315		Epoch:	137		MeanAbsoluteError:

MeanAbsoluteError:	0.0390859991312027		Loss:	0.0025145651223803		Epoch:	140		MeanAbsoluteError:
MeanAbsoluteError:	0.0329549685120583		Loss:	0.0017156039331374		Epoch:	143		MeanAbsoluteError:
MeanAbsoluteError:	0.0256825499236584		Loss:	0.0012017911316694		Epoch:	146		MeanAbsoluteError:
MeanAbsoluteError:	0.0257408283650875		Loss:	0.0009865332477992		Epoch:	149		MeanAbsoluteError:
MeanAbsoluteError:	0.0273976698517799		Loss:	0.0011823049821292		Epoch:	152		MeanAbsoluteError:
MeanAbsoluteError:	0.0285372622311115		Loss:	0.0015433284279425		Epoch:	155		MeanAbsoluteError:
MeanAbsoluteError:	0.0302125662565231		Loss:	0.0021295984292270		Epoch:	158		MeanAbsoluteError:
MeanAbsoluteError:	0.0325568430125713		Loss:	0.0018389636866881		Epoch:	161		MeanAbsoluteError:
MeanAbsoluteError:	0.0288931187242270		Loss:	0.0017507132974320		Epoch:	164		MeanAbsoluteError:
MeanAbsoluteError:	0.0324740521609783		Loss:	0.0016757068307405		Epoch:	167		MeanAbsoluteError:
MeanAbsoluteError:	0.0378320515155792		Loss:	0.0020827587660668		Epoch:	170		MeanAbsoluteError:
MeanAbsoluteError:	0.0293065048754215		Loss:	0.0017131653647604		Epoch:	173		MeanAbsoluteError:
MeanAbsoluteError:	0.0372612513601780		Loss:	0.0022493477112481		Epoch:	176		MeanAbsoluteError:
MeanAbsoluteError:	0.0268558263778687		Loss:	0.0014646025956608		Epoch:	179		MeanAbsoluteError:
MeanAbsoluteError:	0.0308236796408892		Loss:	0.0016497492740330		Epoch:	182		MeanAbsoluteError:
MeanAbsoluteError:	0.0300736334174871		Loss:	0.0017198623889791		Epoch:	185		MeanAbsoluteError:
MeanAbsoluteError:	0.0346332713961601		Loss:	0.0021883776727399		Epoch:	188		MeanAbsoluteError:
MeanAbsoluteError:	0.0291969627141953		Loss:	0.0016912736131677		Epoch:	191		MeanAbsoluteError:
MeanAbsoluteError:	0.0326152071356773		Loss:	0.0023981685101587		Epoch:	194		MeanAbsoluteError:

MeanAbsoluteError: 0.0249930527061224 | Loss: 0.0010144699980239 | Epoch: 40 | MeanAbsoluteError: 0.0251099821180105 | Loss: 0.0009702558995091 | Epoch: 43 | MeanAbsoluteError: 0.0262286253273487 | Loss: 0.0011826954973263 | Epoch: 46 | MeanAbsoluteError: 0.0438615977764130 | Loss: 0.0029156602852579 | Epoch: 49 | MeanAbsoluteError: 0.0396095588803291 | Loss: 0.0024651297634201 | Epoch: 52 | MeanAbsoluteError: 0.0299754813313484 | Loss: 0.0016336921627434 | Epoch: 55 | MeanAbsoluteError: 0.0224273148924112 | Loss: 0.0009723090250710 | Epoch: 58 | MeanAbsoluteError: 0.0243700500577688 | Loss: 0.0010269305452571 | Epoch: 61 | MeanAbsoluteError: 0.0285795219242573 | Loss: 0.0013150820575122 | Epoch: 64 | MeanAbsoluteError: 0.0235760901123285 | Loss: 0.0011584148243336 | Epoch: 67 | MeanAbsoluteError: 0.0281778071075678 | Loss: 0.0011604828967912 | Epoch: 70 | MeanAbsoluteError: 0.0299751330167055 | Loss: 0.0012518782750703 | Epoch: 73 | MeanAbsoluteError: 0.0318193770945072 | Loss: 0.0014749841066077 | Epoch: 76 | MeanAbsoluteError: 0.0298424083739519 | Loss: 0.0013646810964149 | Epoch: 79 | MeanAbsoluteError: 0.0252522882074118 | Loss: 0.0010273767464761 | Epoch: 82 | MeanAbsoluteError: 0.0225144196301699 | Loss: 0.0008793088532652 | Epoch: 85 | MeanAbsoluteError: 0.0266698077321053 | Loss: 0.0014263271919585 | Epoch: 88 | MeanAbsoluteError: 0.0272392034530640 | Loss: 0.0014488329179585 | Epoch: 91 | MeanAbsoluteError: 0.0206654258072376 | Loss: 0.0007099812833725 | Epoch: 94 | MeanAbsoluteError:

MeanAbsoluteError:	0.0314240828156471		Loss:	0.0018612259965656		Epoch:	97		MeanAbsoluteError:	0.0298223216086626
MeanAbsoluteError:	0.0298223216086626		Loss:	0.0014780915475317		Epoch:	100		MeanAbsoluteError:	0.0353239998221397
MeanAbsoluteError:	0.0353239998221397		Loss:	0.0020648369765175		Epoch:	103		MeanAbsoluteError:	0.0367200784385204
MeanAbsoluteError:	0.0367200784385204		Loss:	0.0019954227942175		Epoch:	106		MeanAbsoluteError:	0.0261863321065903
MeanAbsoluteError:	0.0261863321065903		Loss:	0.0013645103234532		Epoch:	109		MeanAbsoluteError:	0.0211539454758167
MeanAbsoluteError:	0.0211539454758167		Loss:	0.0007834805957308		Epoch:	112		MeanAbsoluteError:	0.0280210115015507
MeanAbsoluteError:	0.0280210115015507		Loss:	0.0012407145945222		Epoch:	115		MeanAbsoluteError:	0.0246220156550407
MeanAbsoluteError:	0.0246220156550407		Loss:	0.0009592872562020		Epoch:	118		MeanAbsoluteError:	0.0252812318503857
MeanAbsoluteError:	0.0252812318503857		Loss:	0.0013929698117343		Epoch:	121		MeanAbsoluteError:	0.0269621089100838
MeanAbsoluteError:	0.0269621089100838		Loss:	0.0012997335621289		Epoch:	124		MeanAbsoluteError:	0.0313204675912857
MeanAbsoluteError:	0.0313204675912857		Loss:	0.0015782348512273		Epoch:	127		MeanAbsoluteError:	0.0399994403123856
MeanAbsoluteError:	0.0399994403123856		Loss:	0.0022113237563255		Epoch:	130		MeanAbsoluteError:	0.0443863570690155
MeanAbsoluteError:	0.0443863570690155		Loss:	0.0024649073436324		Epoch:	133		MeanAbsoluteError:	0.0261531621217728
MeanAbsoluteError:	0.0261531621217728		Loss:	0.0010090329409910		Epoch:	136		MeanAbsoluteError:	0.0406115613877773
MeanAbsoluteError:	0.0406115613877773		Loss:	0.0022347073536366		Epoch:	139		MeanAbsoluteError:	0.0314425453543663
MeanAbsoluteError:	0.0314425453543663		Loss:	0.0014564309468759		Epoch:	142		MeanAbsoluteError:	0.0267585404217243
MeanAbsoluteError:	0.0267585404217243		Loss:	0.0010706222113056		Epoch:	145		MeanAbsoluteError:	0.0281627997756004
MeanAbsoluteError:	0.0281627997756004		Loss:	0.0017239008448087		Epoch:	148		MeanAbsoluteError:	0.0293211322277784
MeanAbsoluteError:	0.0293211322277784		Loss:	0.0012720640682216		Epoch:	151		MeanAbsoluteError:	

MeanAbsoluteError:	0.0292988438159227		Loss:	0.0016742575348222		Epoch:	51		MeanAbsoluteError:
MeanAbsoluteError:	0.0344694294035435		Loss:	0.0017996313150174		Epoch:	54		MeanAbsoluteError:
MeanAbsoluteError:	0.0383292846381664		Loss:	0.0024601951174970		Epoch:	57		MeanAbsoluteError:
MeanAbsoluteError:	0.0271987561136484		Loss:	0.0016869087204603		Epoch:	60		MeanAbsoluteError:
MeanAbsoluteError:	0.0334605798125267		Loss:	0.0023148667844777		Epoch:	63		MeanAbsoluteError:
MeanAbsoluteError:	0.0414203368127346		Loss:	0.0029072683204764		Epoch:	66		MeanAbsoluteError:
MeanAbsoluteError:	0.0274894684553146		Loss:	0.0011830580562154		Epoch:	69		MeanAbsoluteError:
MeanAbsoluteError:	0.0293164085596800		Loss:	0.0016413335522105		Epoch:	72		MeanAbsoluteError:
MeanAbsoluteError:	0.0290586519986391		Loss:	0.0015197330836340		Epoch:	75		MeanAbsoluteError:
MeanAbsoluteError:	0.0408562645316124		Loss:	0.0024491792012538		Epoch:	78		MeanAbsoluteError:
MeanAbsoluteError:	0.0257311239838600		Loss:	0.0015414308041467		Epoch:	81		MeanAbsoluteError:
MeanAbsoluteError:	0.0302357487380505		Loss:	0.0018701249168121		Epoch:	84		MeanAbsoluteError:
MeanAbsoluteError:	0.0294500961899757		Loss:	0.0015102081399943		Epoch:	87		MeanAbsoluteError:
MeanAbsoluteError:	0.0280338637530804		Loss:	0.0012094952398911		Epoch:	90		MeanAbsoluteError:
MeanAbsoluteError:	0.0361978784203529		Loss:	0.0023981324962473		Epoch:	93		MeanAbsoluteError:
MeanAbsoluteError:	0.0302943512797356		Loss:	0.0014693043194711		Epoch:	96		MeanAbsoluteError:
MeanAbsoluteError:	0.0345915183424950		Loss:	0.0020450413576327		Epoch:	99		MeanAbsoluteError:
MeanAbsoluteError:	0.0326125919818878		Loss:	0.0018085532605515		Epoch:	102		MeanAbsoluteError:
MeanAbsoluteError:	0.0254972148686647		Loss:	0.0017424789174194		Epoch:	105		MeanAbsoluteError:

MeanAbsoluteError:	0.03046511111364365		Loss:	0.0015967655344866		Epoch:	108		MeanAbsoluteError:
MeanAbsoluteError:	0.0253878347575665		Loss:	0.0010056156731610		Epoch:	111		MeanAbsoluteError:
MeanAbsoluteError:	0.0347823873162270		Loss:	0.0019821137289650		Epoch:	114		MeanAbsoluteError:
MeanAbsoluteError:	0.0435186065733433		Loss:	0.0035459363134578		Epoch:	117		MeanAbsoluteError:
MeanAbsoluteError:	0.0342162661254406		Loss:	0.0018513255885669		Epoch:	120		MeanAbsoluteError:
MeanAbsoluteError:	0.0295951347798109		Loss:	0.0018714041715222		Epoch:	123		MeanAbsoluteError:
MeanAbsoluteError:	0.0376754067838192		Loss:	0.0025903295609169		Epoch:	126		MeanAbsoluteError:
MeanAbsoluteError:	0.0304889697581530		Loss:	0.0016689065877082		Epoch:	129		MeanAbsoluteError:
MeanAbsoluteError:	0.0287471488118172		Loss:	0.0012772462879574		Epoch:	132		MeanAbsoluteError:
MeanAbsoluteError:	0.0300400611013174		Loss:	0.0018377007051770		Epoch:	135		MeanAbsoluteError:
MeanAbsoluteError:	0.0376685969531536		Loss:	0.0022182437242009		Epoch:	138		MeanAbsoluteError:
MeanAbsoluteError:	0.0281013678759336		Loss:	0.0016442738914131		Epoch:	141		MeanAbsoluteError:
MeanAbsoluteError:	0.0283485148102045		Loss:	0.0014933777524025		Epoch:	144		MeanAbsoluteError:
MeanAbsoluteError:	0.0333470366895199		Loss:	0.0019464590420414		Epoch:	147		MeanAbsoluteError:
MeanAbsoluteError:	0.0344715379178524		Loss:	0.0020864511773522		Epoch:	150		MeanAbsoluteError:
MeanAbsoluteError:	0.0349323675036430		Loss:	0.0021382904877620		Epoch:	153		MeanAbsoluteError:
MeanAbsoluteError:	0.0376243442296982		Loss:	0.0018828275081302		Epoch:	156		MeanAbsoluteError:
MeanAbsoluteError:	0.0244383811950684		Loss:	0.0010395194071212		Epoch:	159		MeanAbsoluteError:
MeanAbsoluteError:	0.0296687837690115		Loss:	0.0016900802480190		Epoch:	162		MeanAbsoluteError:

MeanAbsoluteError:	0.0352939553558826		Loss:	0.0018103357516728		Epoch:	45		MeanAbsoluteE			
MeanAbsoluteError:	0.0284352879971266		Loss:	0.0012252627722254		Epoch:	48		MeanAbsoluteE			
MeanAbsoluteError:	0.0251844469457865		Loss:	0.0010939955370434		Epoch:	51		MeanAbsoluteE			
MeanAbsoluteError:	0.0298968367278576		Loss:	0.0018608756862315		Epoch:	54		MeanAbsoluteE			
MeanAbsoluteError:	0.0280760470777750		Loss:	0.0014979041048459		Epoch:	57		MeanAbsoluteE			
MeanAbsoluteError:	0.0274460464715958		Loss:	0.0011714347970805		Epoch:	60		MeanAbsoluteE			
MeanAbsoluteError:	0.0281603559851646		Loss:	0.0013163296888316		Epoch:	63		MeanAbsoluteE			
MeanAbsoluteError:	0.0241794753819704		Loss:	0.0009317000014042		Epoch:	66		MeanAbsoluteE			
MeanAbsoluteError:	0.0223099216818810		Loss:	0.0008708491597125		Epoch:	69		MeanAbsoluteE			
MeanAbsoluteError:	0.0373122505843639		Loss:	0.0021633775738467		Epoch:	72		MeanAbsoluteE			
MeanAbsoluteError:	0.0317080989480019		Loss:	0.0016941127999287		Epoch:	75		MeanAbsoluteE			
MeanAbsoluteError:	0.0242307074368000		Loss:	0.0009790996498071		Epoch:	78		MeanAbsoluteE			
MeanAbsoluteError:	0.0369772352278233		Loss:	0.0023148058431356		Epoch:	81		MeanAbsoluteE			
MeanAbsoluteError:	0.0315312035381794		Loss:	0.0017068094873269		Epoch:	84		MeanAbsoluteE			
MeanAbsoluteError:	0.0238929726183414		Loss:	0.0010959014096963		Epoch:	87		MeanAbsoluteE			
MeanAbsoluteError:	0.0306991823017597		Loss:	0.0017142851886872		Epoch:	90		MeanAbsoluteE			
Epoch:	92		MeanAbsoluteError:	0.0285295508801937		Loss:	0.0020459948573261		Epoch:	93		M
MeanAbsoluteError:	0.0268037989735603		Loss:	0.0012391417403705		Epoch:	95		MeanAbsoluteE			
MeanAbsoluteError:	0.0272518377751112		Loss:	0.0011845303566328		Epoch:	98		MeanAbsoluteE			

MeanAbsoluteError:	0.0261498037725687		Loss:	0.0010047240877092		Epoch:	101		MeanAbsoluteError:	0.0261498037725687
MeanAbsoluteError:	0.0294485781341791		Loss:	0.0015578188052002		Epoch:	104		MeanAbsoluteError:	0.0294485781341791
MeanAbsoluteError:	0.0263538174331188		Loss:	0.0011917541851290		Epoch:	107		MeanAbsoluteError:	0.0263538174331188
MeanAbsoluteError:	0.0257081892341375		Loss:	0.0017346564480769		Epoch:	110		MeanAbsoluteError:	0.0257081892341375
MeanAbsoluteError:	0.0272347442805767		Loss:	0.0010779483626331		Epoch:	113		MeanAbsoluteError:	0.0272347442805767
MeanAbsoluteError:	0.0307241417467594		Loss:	0.0018262493748417		Epoch:	116		MeanAbsoluteError:	0.0307241417467594
MeanAbsoluteError:	0.0305989626795053		Loss:	0.0016088827313589		Epoch:	119		MeanAbsoluteError:	0.0305989626795053
MeanAbsoluteError:	0.0290838461369276		Loss:	0.0014253464427644		Epoch:	122		MeanAbsoluteError:	0.0290838461369276
MeanAbsoluteError:	0.0305249635130167		Loss:	0.0014093329331705		Epoch:	125		MeanAbsoluteError:	0.0305249635130167
MeanAbsoluteError:	0.0429239161312580		Loss:	0.0026849085198981		Epoch:	128		MeanAbsoluteError:	0.0429239161312580
MeanAbsoluteError:	0.0354175195097923		Loss:	0.0022111128949161		Epoch:	131		MeanAbsoluteError:	0.0354175195097923
MeanAbsoluteError:	0.0233202725648880		Loss:	0.0009231948642991		Epoch:	134		MeanAbsoluteError:	0.0233202725648880
MeanAbsoluteError:	0.0291329473257065		Loss:	0.0014539917881068		Epoch:	137		MeanAbsoluteError:	0.0291329473257065
MeanAbsoluteError:	0.0299759618937969		Loss:	0.0015024646085554		Epoch:	140		MeanAbsoluteError:	0.0299759618937969
MeanAbsoluteError:	0.0339423827826977		Loss:	0.0019400311700468		Epoch:	143		MeanAbsoluteError:	0.0339423827826977
MeanAbsoluteError:	0.0329283699393272		Loss:	0.0016358881673243		Epoch:	146		MeanAbsoluteError:	0.0329283699393272
MeanAbsoluteError:	0.0361309349536896		Loss:	0.0017477092541022		Epoch:	149		MeanAbsoluteError:	0.0361309349536896
MeanAbsoluteError:	0.0297470763325691		Loss:	0.0019486354347984		Epoch:	152		MeanAbsoluteError:	0.0297470763325691
MeanAbsoluteError:	0.0236697979271412		Loss:	0.0010236651620029		Epoch:	155		MeanAbsoluteError:	0.0236697979271412

```
MeanAbsoluteError: 0.0242082849144936 | Loss: 0.0009647943453664 | Epoch: 158 | MeanAbsoluteE  

MeanAbsoluteError: 0.0328540094196796 | Loss: 0.0017647102940828 | Epoch: 161 | MeanAbsoluteE  

MeanAbsoluteError: 0.0274898000061512 | Loss: 0.0010949278948829 | Epoch: 164 | MeanAbsoluteE  

MeanAbsoluteError: 0.0275878421962261 | Loss: 0.0012728944503968 | Epoch: 167 | MeanAbsoluteE  

MeanAbsoluteError: 0.0401649139821529 | Loss: 0.0022107369732112 | Epoch: 170 | MeanAbsoluteE  

MeanAbsoluteError: 0.0250850152224302 | Loss: 0.0010277686898397 | Epoch: 173 | MeanAbsoluteE  

MeanAbsoluteError: 0.0354162454605103 | Loss: 0.0020162078851302 | Epoch: 176 | MeanAbsoluteE  

MeanAbsoluteError: 0.0371210724115372 | Loss: 0.0021990797582215 | Epoch: 179 | MeanAbsoluteE  

MeanAbsoluteError: 0.0294383335858583 | Loss: 0.0013311786335959 | Epoch: 182 | MeanAbsoluteE  

MeanAbsoluteError: 0.0270031206309795 | Loss: 0.0015558541344944 | Epoch: 185 | MeanAbsoluteE  

MeanAbsoluteError: 0.0351253636181355 | Loss: 0.0019716245920530 | Epoch: 188 | MeanAbsoluteE  

MeanAbsoluteError: 0.0262540858238935 | Loss: 0.0011104165875752 | Epoch: 191 | MeanAbsoluteE  

MeanAbsoluteError: 0.0267670694738626 | Loss: 0.0011589729692787 | Early stopping at epoch 19  

Fold: 6  

Epoch: 1 | MeanAbsoluteError: 0.1028370559215546 | Loss: 0.0189826919564179 | Epoch: 2 | Mean  

MeanAbsoluteError: 0.0730783268809319 | Loss: 0.0077703818678856 | Epoch: 4 | MeanAbsoluteErr  

MeanAbsoluteError: 0.0541634075343609 | Loss: 0.0054955877962389 | Epoch: 7 | MeanAbsoluteErr  

MeanAbsoluteError: 0.0571859553456306 | Loss: 0.0051246076057266 | Epoch: 10 | MeanAbsoluteE  

MeanAbsoluteError: 0.0584732182323933 | Loss: 0.0064742517923670 | Epoch: 13 | MeanAbsoluteE  

MeanAbsoluteError: 0.0597118698060513 | Loss: 0.0052666349802166 | Epoch: 16 | MeanAbsoluteE
```

MeanAbsoluteError:	0.0407193191349506		Loss:	0.0040315325438444		Epoch:	19		MeanAbsoluteError:
MeanAbsoluteError:	0.0433678664267063		Loss:	0.0034537192633642		Epoch:	22		MeanAbsoluteError:
MeanAbsoluteError:	0.0321137681603432		Loss:	0.0016297865492691		Epoch:	25		MeanAbsoluteError:
MeanAbsoluteError:	0.0529956147074699		Loss:	0.0042341853758054		Epoch:	28		MeanAbsoluteError:
MeanAbsoluteError:	0.0385552681982517		Loss:	0.0025208613702229		Epoch:	31		MeanAbsoluteError:
MeanAbsoluteError:	0.0331960730254650		Loss:	0.0022056117470908		Epoch:	34		MeanAbsoluteError:
MeanAbsoluteError:	0.0340473838150501		Loss:	0.0017903291404114		Epoch:	37		MeanAbsoluteError:
MeanAbsoluteError:	0.0370962321758270		Loss:	0.0023454168695025		Epoch:	40		MeanAbsoluteError:
MeanAbsoluteError:	0.0323405414819717		Loss:	0.0015592446579831		Epoch:	43		MeanAbsoluteError:
MeanAbsoluteError:	0.0403276234865189		Loss:	0.0023796391074679		Epoch:	46		MeanAbsoluteError:
MeanAbsoluteError:	0.0362185090780258		Loss:	0.0021589308245374		Epoch:	49		MeanAbsoluteError:
MeanAbsoluteError:	0.0334958173334599		Loss:	0.0018819864365339		Epoch:	52		MeanAbsoluteError:
MeanAbsoluteError:	0.0347799882292747		Loss:	0.0020186389280882		Epoch:	55		MeanAbsoluteError:
MeanAbsoluteError:	0.0372370518743992		Loss:	0.0024388381280005		Epoch:	58		MeanAbsoluteError:
MeanAbsoluteError:	0.0405829250812531		Loss:	0.0023337913943189		Epoch:	61		MeanAbsoluteError:
MeanAbsoluteError:	0.0294870585203171		Loss:	0.0015158530981613		Epoch:	64		MeanAbsoluteError:
MeanAbsoluteError:	0.0291451755911112		Loss:	0.0022166290935794		Epoch:	67		MeanAbsoluteError:
MeanAbsoluteError:	0.0346508063375950		Loss:	0.0027566413212168		Epoch:	70		MeanAbsoluteError:
MeanAbsoluteError:	0.0294269677251577		Loss:	0.0014460050525875		Epoch:	73		MeanAbsoluteError:

```

MeanAbsoluteError: 0.0311381388455629 | Loss: 0.0018681905598247 | Epoch: 76 | MeanAbsoluteError: 0.0311381388455629
MeanAbsoluteError: 0.0325682945549488 | Loss: 0.0019905739422289 | Epoch: 79 | MeanAbsoluteError: 0.0325682945549488
MeanAbsoluteError: 0.0351242981851101 | Loss: 0.0019987939823685 | Epoch: 82 | MeanAbsoluteError: 0.0351242981851101
MeanAbsoluteError: 0.0333644300699234 | Loss: 0.0024006119430331 | Epoch: 85 | MeanAbsoluteError: 0.0333644300699234
MeanAbsoluteError: 0.0281554814428091 | Loss: 0.0017851170351995 | Epoch: 88 | MeanAbsoluteError: 0.0281554814428091
MeanAbsoluteError: 0.0264260433614254 | Loss: 0.0015716295539668 | Epoch: 91 | MeanAbsoluteError: 0.0264260433614254
MeanAbsoluteError: 0.0326556190848351 | Loss: 0.0021936519104721 | Epoch: 94 | MeanAbsoluteError: 0.0326556190848351
MeanAbsoluteError: 0.0354536920785904 | Loss: 0.0021076756487933 | Epoch: 97 | MeanAbsoluteError: 0.0354536920785904
MeanAbsoluteError: 0.0400020591914654 | Loss: 0.0031318958582623 | Epoch: 100 | MeanAbsoluteError: 0.0400020591914654
MeanAbsoluteError: 0.0398079268634319 | Loss: 0.0026936401346964 | Epoch: 103 | MeanAbsoluteError: 0.0398079268634319
MeanAbsoluteError: 0.0291024949401617 | Loss: 0.0016031165474227 | Epoch: 106 | MeanAbsoluteError: 0.0291024949401617
MeanAbsoluteError: 0.0361901819705963 | Loss: 0.0023243541197319 | Epoch: 109 | MeanAbsoluteError: 0.0361901819705963
MeanAbsoluteError: 0.0379751436412334 | Loss: 0.0023013421783357 | Epoch: 112 | MeanAbsoluteError: 0.0379751436412334
MeanAbsoluteError: 0.0294586662203074 | Loss: 0.0019976706826128 | Epoch: 115 | MeanAbsoluteError: 0.0294586662203074
MeanAbsoluteError: 0.0287080295383930 | Loss: 0.0015366281981447 | Epoch: 118 | MeanAbsoluteError: 0.0287080295383930
MeanAbsoluteError: 0.0324073731899261 | Loss: 0.0019265545249384 | Early stopping at epoch 119
Fold: 7
Epoch: 1 | MeanAbsoluteError: 0.1174591258168221 | Loss: 0.0215794603739466 | Epoch: 2 | MeanAbsoluteError: 0.0725061818957329
MeanAbsoluteError: 0.0725061818957329 | Loss: 0.0083495392464101 | Epoch: 4 | MeanAbsoluteError: 0.0725061818957329
MeanAbsoluteError: 0.0580258369445801 | Loss: 0.0060746019672869 | Epoch: 7 | MeanAbsoluteError: 0.0580258369445801

```

MeanAbsoluteError: 0.0505628101527691		Loss: 0.0061615798622370		Epoch: 10		MeanAbsoluteError: 0.0448244065046310
MeanAbsoluteError: 0.0448244065046310		Loss: 0.0034130595491401		Epoch: 13		MeanAbsoluteError: 0.0438367128372192
MeanAbsoluteError: 0.0438367128372192		Loss: 0.0030883520874860		Epoch: 16		MeanAbsoluteError: 0.0433518886566162
MeanAbsoluteError: 0.0433518886566162		Loss: 0.0026693870853965		Epoch: 19		MeanAbsoluteError: 0.0307148937135935
MeanAbsoluteError: 0.0307148937135935		Loss: 0.0018067969566411		Epoch: 22		MeanAbsoluteError: 0.0330751761794090
MeanAbsoluteError: 0.0330751761794090		Loss: 0.0019466363592073		Epoch: 25		MeanAbsoluteError: 0.0387557372450829
MeanAbsoluteError: 0.0387557372450829		Loss: 0.0024290077022410		Epoch: 28		MeanAbsoluteError: 0.0408230200409889
MeanAbsoluteError: 0.0408230200409889		Loss: 0.0026646044903568		Epoch: 31		MeanAbsoluteError: 0.0296457633376122
MeanAbsoluteError: 0.0296457633376122		Loss: 0.0014179092249833		Epoch: 34		MeanAbsoluteError: 0.0324876196682453
MeanAbsoluteError: 0.0324876196682453		Loss: 0.0021233694195481		Epoch: 37		MeanAbsoluteError: 0.0246996767818928
MeanAbsoluteError: 0.0246996767818928		Loss: 0.0010544431362567		Epoch: 40		MeanAbsoluteError: 0.0242969579994678
MeanAbsoluteError: 0.0242969579994678		Loss: 0.0009511349489912		Epoch: 43		MeanAbsoluteError: 0.0295211765915155
MeanAbsoluteError: 0.0295211765915155		Loss: 0.0014828191737511		Epoch: 46		MeanAbsoluteError: 0.0298874489963055
MeanAbsoluteError: 0.0298874489963055		Loss: 0.0014705188389468		Epoch: 49		MeanAbsoluteError: 0.0236435849219561
MeanAbsoluteError: 0.0236435849219561		Loss: 0.0008636958123783		Epoch: 52		MeanAbsoluteError: 0.0283979102969170
MeanAbsoluteError: 0.0283979102969170		Loss: 0.0012592080945199		Epoch: 55		MeanAbsoluteError: 0.0399386100471020
MeanAbsoluteError: 0.0399386100471020		Loss: 0.0028325327938156		Epoch: 58		MeanAbsoluteError: 0.0239721424877644
MeanAbsoluteError: 0.0239721424877644		Loss: 0.0010542349025075		Epoch: 61		MeanAbsoluteError: 0.0281397774815559
MeanAbsoluteError: 0.0281397774815559		Loss: 0.0013736135276434		Epoch: 64		

MeanAbsoluteError:	0.0275461729615927		Loss:	0.0015766271057406		Epoch:	67		MeanAbsoluteError:	0.0275461729615927
MeanAbsoluteError:	0.0285801608115435		Loss:	0.0014900661023733		Epoch:	70		MeanAbsoluteError:	0.0285801608115435
MeanAbsoluteError:	0.0376192927360535		Loss:	0.0024416336257543		Epoch:	73		MeanAbsoluteError:	0.0376192927360535
MeanAbsoluteError:	0.0204386282712221		Loss:	0.0010946806058720		Epoch:	76		MeanAbsoluteError:	0.0204386282712221
MeanAbsoluteError:	0.0313175842165947		Loss:	0.0015736887830177		Epoch:	79		MeanAbsoluteError:	0.0313175842165947
MeanAbsoluteError:	0.0193850714713335		Loss:	0.0007569501259630		Epoch:	82		MeanAbsoluteError:	0.0193850714713335
MeanAbsoluteError:	0.0259185861796141		Loss:	0.0014248968509492		Epoch:	85		MeanAbsoluteError:	0.0259185861796141
MeanAbsoluteError:	0.0266274064779282		Loss:	0.0012173502057392		Epoch:	88		MeanAbsoluteError:	0.0266274064779282
MeanAbsoluteError:	0.0231321807950735		Loss:	0.0011647666937539		Epoch:	91		MeanAbsoluteError:	0.0231321807950735
MeanAbsoluteError:	0.0222006682306528		Loss:	0.0008564812389003		Epoch:	94		MeanAbsoluteError:	0.0222006682306528
MeanAbsoluteError:	0.0349811688065529		Loss:	0.0025985566233950		Epoch:	97		MeanAbsoluteError:	0.0349811688065529
MeanAbsoluteError:	0.0295862276107073		Loss:	0.0014653405274398		Epoch:	100		MeanAbsoluteError:	0.0295862276107073
MeanAbsoluteError:	0.0267510525882244		Loss:	0.0013189973376159		Epoch:	103		MeanAbsoluteError:	0.0267510525882244
MeanAbsoluteError:	0.0261370260268450		Loss:	0.0010922479393360		Epoch:	106		MeanAbsoluteError:	0.0261370260268450
MeanAbsoluteError:	0.0335521847009659		Loss:	0.0017650537026514		Epoch:	109		MeanAbsoluteError:	0.0335521847009659
MeanAbsoluteError:	0.0253927595913410		Loss:	0.0011945350187099		Epoch:	112		MeanAbsoluteError:	0.0253927595913410
MeanAbsoluteError:	0.0343814492225647		Loss:	0.0017271646897175		Epoch:	115		MeanAbsoluteError:	0.0343814492225647
MeanAbsoluteError:	0.0272312909364700		Loss:	0.0011196138387147		Epoch:	118		MeanAbsoluteError:	0.0272312909364700
MeanAbsoluteError:	0.0218562707304955		Loss:	0.0008298703729192		Epoch:	121		MeanAbsoluteError:	0.0218562707304955

MeanAbsoluteError: 0.0292379707098007		Loss: 0.0013136408358280		Epoch: 124		MeanAbsoluteError: 0.0292379707098007
MeanAbsoluteError: 0.0308411940932274		Loss: 0.0018772997261424		Epoch: 127		MeanAbsoluteError: 0.0308411940932274
MeanAbsoluteError: 0.0238788072019815		Loss: 0.0011467014015320		Epoch: 130		MeanAbsoluteError: 0.0238788072019815
MeanAbsoluteError: 0.0237683970481157		Loss: 0.0011247993263948		Epoch: 133		MeanAbsoluteError: 0.0237683970481157
MeanAbsoluteError: 0.0305814743041992		Loss: 0.0014226760249585		Epoch: 136		MeanAbsoluteError: 0.0305814743041992
MeanAbsoluteError: 0.0198193117976189		Loss: 0.0008946008456405		Epoch: 139		MeanAbsoluteError: 0.0198193117976189
MeanAbsoluteError: 0.0321757681667805		Loss: 0.0015287006806050		Epoch: 142		MeanAbsoluteError: 0.0321757681667805
MeanAbsoluteError: 0.0289467126131058		Loss: 0.0013980117385342		Epoch: 145		MeanAbsoluteError: 0.0289467126131058
MeanAbsoluteError: 0.0311226136982441		Loss: 0.0015265830526395		Epoch: 148		MeanAbsoluteError: 0.0311226136982441
MeanAbsoluteError: 0.0242322068661451		Loss: 0.0008450232718522		Epoch: 151		MeanAbsoluteError: 0.0242322068661451
MeanAbsoluteError: 0.0356973186135292		Loss: 0.0020147853730513		Epoch: 154		MeanAbsoluteError: 0.0356973186135292
MeanAbsoluteError: 0.0319712385535240		Loss: 0.0015889259562495		Epoch: 157		MeanAbsoluteError: 0.0319712385535240
MeanAbsoluteError: 0.0293604414910078		Loss: 0.0017947190956745		Epoch: 160		MeanAbsoluteError: 0.0293604414910078
MeanAbsoluteError: 0.0219196062535048		Loss: 0.0010129397352492		Epoch: 163		MeanAbsoluteError: 0.0219196062535048
MeanAbsoluteError: 0.0228972211480141		Loss: 0.0009871731412464		Epoch: 166		MeanAbsoluteError: 0.0228972211480141
MeanAbsoluteError: 0.0233855862170458		Loss: 0.0013260879329339		Epoch: 169		MeanAbsoluteError: 0.0233855862170458
MeanAbsoluteError: 0.0248810313642025		Loss: 0.0009505722139563		Epoch: 172		MeanAbsoluteError: 0.0248810313642025
MeanAbsoluteError: 0.0325686670839787		Loss: 0.0018402921268716		Epoch: 175		MeanAbsoluteError: 0.0325686670839787
MeanAbsoluteError: 0.0351037271320820		Loss: 0.0022394290426746		Epoch: 178		MeanAbsoluteError: 0.0351037271320820

MeanAbsoluteError:	0.0346933789551258		Loss:	0.0023100258632829		Epoch:	37		MeanAbsoluteError:
MeanAbsoluteError:	0.0399653688073158		Loss:	0.0021337465150282		Epoch:	40		MeanAbsoluteError:
MeanAbsoluteError:	0.0356065146625042		Loss:	0.0018622146869477		Epoch:	43		MeanAbsoluteError:
MeanAbsoluteError:	0.0307816751301289		Loss:	0.0014445366105065		Epoch:	46		MeanAbsoluteError:
MeanAbsoluteError:	0.0283598192036152		Loss:	0.0014017424296721		Epoch:	49		MeanAbsoluteError:
MeanAbsoluteError:	0.0324849300086498		Loss:	0.0017646171618253		Epoch:	52		MeanAbsoluteError:
MeanAbsoluteError:	0.0469261966645718		Loss:	0.0032232729052859		Epoch:	55		MeanAbsoluteError:
MeanAbsoluteError:	0.0264582559466362		Loss:	0.0011580454967251		Epoch:	58		MeanAbsoluteError:
MeanAbsoluteError:	0.0307426862418652		Loss:	0.0014760141743214		Epoch:	61		MeanAbsoluteError:
MeanAbsoluteError:	0.0287805031985044		Loss:	0.0016404410458303		Epoch:	64		MeanAbsoluteError:
MeanAbsoluteError:	0.0305327959358692		Loss:	0.0021024536649098		Epoch:	67		MeanAbsoluteError:
MeanAbsoluteError:	0.0274719838052988		Loss:	0.0012445551692508		Epoch:	70		MeanAbsoluteError:
MeanAbsoluteError:	0.0292349401861429		Loss:	0.0015050791220606		Epoch:	73		MeanAbsoluteError:
MeanAbsoluteError:	0.0254366751760244		Loss:	0.0012096374578375		Epoch:	76		MeanAbsoluteError:
MeanAbsoluteError:	0.0281493570655584		Loss:	0.0016932720651052		Epoch:	79		MeanAbsoluteError:
MeanAbsoluteError:	0.0340066514909267		Loss:	0.0022448368337271		Epoch:	82		MeanAbsoluteError:
MeanAbsoluteError:	0.0317698568105698		Loss:	0.0016033966593178		Epoch:	85		MeanAbsoluteError:
MeanAbsoluteError:	0.0253509711474180		Loss:	0.0015309218683147		Epoch:	88		MeanAbsoluteError:
MeanAbsoluteError:	0.0404447652399540		Loss:	0.0023162827960082		Epoch:	91		MeanAbsoluteError:

MeanAbsoluteError:	0.0286603495478630		Loss:	0.0011921854290579		Epoch:	94		MeanAbsoluteError:	0.0286603495478630
MeanAbsoluteError:	0.0310556404292583		Loss:	0.0015590530154960		Epoch:	97		MeanAbsoluteError:	0.0310556404292583
MeanAbsoluteError:	0.0263911131769419		Loss:	0.0013995090848766		Epoch:	100		MeanAbsoluteError:	0.0263911131769419
MeanAbsoluteError:	0.0270922593772411		Loss:	0.0013612425952618		Epoch:	103		MeanAbsoluteError:	0.0270922593772411
MeanAbsoluteError:	0.0262554138898849		Loss:	0.0012419440359476		Epoch:	106		MeanAbsoluteError:	0.0262554138898849
MeanAbsoluteError:	0.0281315129250288		Loss:	0.0011817111039168		Epoch:	109		MeanAbsoluteError:	0.0281315129250288
MeanAbsoluteError:	0.0275117158889771		Loss:	0.0015621238105398		Epoch:	112		MeanAbsoluteError:	0.0275117158889771
MeanAbsoluteError:	0.0230747796595097		Loss:	0.0012357860728766		Epoch:	115		MeanAbsoluteError:	0.0230747796595097
MeanAbsoluteError:	0.0296441912651062		Loss:	0.0013454349245876		Epoch:	118		MeanAbsoluteError:	0.0296441912651062
MeanAbsoluteError:	0.0220996458083391		Loss:	0.0011975387169514		Epoch:	121		MeanAbsoluteError:	0.0220996458083391
MeanAbsoluteError:	0.0346966534852982		Loss:	0.0018925505490708		Epoch:	124		MeanAbsoluteError:	0.0346966534852982
MeanAbsoluteError:	0.0262452587485313		Loss:	0.0015782864523187		Epoch:	127		MeanAbsoluteError:	0.0262452587485313
MeanAbsoluteError:	0.0321820154786110		Loss:	0.0015364465070888		Epoch:	130		MeanAbsoluteError:	0.0321820154786110
MeanAbsoluteError:	0.0259668044745922		Loss:	0.0008883182474944		Epoch:	133		MeanAbsoluteError:	0.0259668044745922
MeanAbsoluteError:	0.0217172168195248		Loss:	0.0007353102389191		Epoch:	136		MeanAbsoluteError:	0.0217172168195248
MeanAbsoluteError:	0.0274241063743830		Loss:	0.0013179566594772		Epoch:	139		MeanAbsoluteError:	0.0274241063743830
MeanAbsoluteError:	0.0297271069139242		Loss:	0.0017953635625807		Epoch:	142		MeanAbsoluteError:	0.0297271069139242
MeanAbsoluteError:	0.0274721905589104		Loss:	0.0016685585641036		Epoch:	145		MeanAbsoluteError:	0.0274721905589104
MeanAbsoluteError:	0.0344022177159786		Loss:	0.0019372985781437		Epoch:	148		MeanAbsoluteError:	0.0344022177159786

MeanAbsoluteError:	0.0224078278988600		Loss:	0.0008832589041309		Epoch:	151		MeanAbsoluteError:
MeanAbsoluteError:	0.0203842949122190		Loss:	0.0007573440587813		Epoch:	154		MeanAbsoluteError:
MeanAbsoluteError:	0.0301984604448080		Loss:	0.0015621956720549		Epoch:	157		MeanAbsoluteError:
MeanAbsoluteError:	0.0363770723342896		Loss:	0.0027440023675029		Epoch:	160		MeanAbsoluteError:
MeanAbsoluteError:	0.0262810755521059		Loss:	0.0013558502126086		Epoch:	163		MeanAbsoluteError:
MeanAbsoluteError:	0.0230202693492174		Loss:	0.0009072318747972		Epoch:	166		MeanAbsoluteError:
MeanAbsoluteError:	0.0243662595748901		Loss:	0.0011882648364657		Epoch:	169		MeanAbsoluteError:
MeanAbsoluteError:	0.0241734981536865		Loss:	0.0011895203164646		Epoch:	172		MeanAbsoluteError:
MeanAbsoluteError:	0.0224750414490700		Loss:	0.0008414583031221		Epoch:	175		MeanAbsoluteError:
MeanAbsoluteError:	0.0273708216845989		Loss:	0.0016629685997032		Epoch:	178		MeanAbsoluteError:
MeanAbsoluteError:	0.0240628886967897		Loss:	0.0011398097101067		Epoch:	181		MeanAbsoluteError:
MeanAbsoluteError:	0.0304818060249090		Loss:	0.0016605072471845		Epoch:	184		MeanAbsoluteError:
MeanAbsoluteError:	0.0235693734139204		Loss:	0.0009936908276619		Epoch:	187		MeanAbsoluteError:
MeanAbsoluteError:	0.0310732219368219		Loss:	0.0015977966333074		Epoch:	190		MeanAbsoluteError:
MeanAbsoluteError:	0.0305306427180767		Loss:	0.0013920936949684		Epoch:	193		MeanAbsoluteError:
MeanAbsoluteError:	0.0278504379093647		Loss:	0.0012161165692045		Epoch:	196		MeanAbsoluteError:
MeanAbsoluteError:	0.0259171202778816		Loss:	0.0019033498787654		Epoch:	199		MeanAbsoluteError:
MeanAbsoluteError:	0.0252894349396229		Loss:	0.0010597275686450		Epoch:	202		MeanAbsoluteError:
MeanAbsoluteError:	0.0249080434441566		Loss:	0.0010012197807165		Epoch:	205		MeanAbsoluteError:

MeanAbsoluteError:	0.0282842610031366		Loss:	0.0013977631710337		Epoch:	208		MeanAbsoluteError:
MeanAbsoluteError:	0.0311016794294119		Loss:	0.0015652943069914		Epoch:	211		MeanAbsoluteError:
MeanAbsoluteError:	0.0257380455732346		Loss:	0.0010933840946694		Epoch:	214		MeanAbsoluteError:
MeanAbsoluteError:	0.0207174494862556		Loss:	0.0006821846639338		Epoch:	217		MeanAbsoluteError:
MeanAbsoluteError:	0.0220520161092281		Loss:	0.0007534899757177		Epoch:	220		MeanAbsoluteError:
MeanAbsoluteError:	0.0282060839235783		Loss:	0.0013894794226092		Epoch:	223		MeanAbsoluteError:
MeanAbsoluteError:	0.0261391140520573		Loss:	0.0011627830681391		Epoch:	226		MeanAbsoluteError:
MeanAbsoluteError:	0.0232427045702934		Loss:	0.0009405037687559		Epoch:	229		MeanAbsoluteError:
MeanAbsoluteError:	0.0276988595724106		Loss:	0.0011058919065233		Epoch:	232		MeanAbsoluteError:
MeanAbsoluteError:	0.0270410068333149		Loss:	0.0014406097720244		Epoch:	235		MeanAbsoluteError:
MeanAbsoluteError:	0.0230942126363516		Loss:	0.0010753719876188		Epoch:	238		MeanAbsoluteError:
MeanAbsoluteError:	0.0317869707942009		Loss:	0.0032090357770877		Epoch:	241		MeanAbsoluteError:
MeanAbsoluteError:	0.0331509225070477		Loss:	0.0016816884245990		Epoch:	244		MeanAbsoluteError:
MeanAbsoluteError:	0.0297519881278276		Loss:	0.0017075802981188		Epoch:	247		MeanAbsoluteError:
MeanAbsoluteError:	0.0290315002202988		Loss:	0.0016008706297725		Epoch:	250		MeanAbsoluteError:
MeanAbsoluteError:	0.0369610227644444		Loss:	0.0020049523113162		Epoch:	253		MeanAbsoluteError:
MeanAbsoluteError:	0.0272037573158741		Loss:	0.0012390113967870		Epoch:	256		MeanAbsoluteError:
MeanAbsoluteError:	0.0231915991753340		Loss:	0.0012573532793405		Epoch:	259		MeanAbsoluteError:
MeanAbsoluteError:	0.0264076646417379		Loss:	0.0011957478751096		Epoch:	262		MeanAbsoluteError:

[illegible]

MeanAbsoluteError:	0.0288332104682922		Loss:	0.0015842295667556		Epoch:	39		MeanAbsoluteError:	0.0288332104682922
MeanAbsoluteError:	0.0295581128448248		Loss:	0.0016537569629561		Epoch:	42		MeanAbsoluteError:	0.0295581128448248
MeanAbsoluteError:	0.0336967259645462		Loss:	0.0024129396437534		Epoch:	45		MeanAbsoluteError:	0.0336967259645462
MeanAbsoluteError:	0.0320559777319431		Loss:	0.0026454876642674		Epoch:	48		MeanAbsoluteError:	0.0320559777319431
MeanAbsoluteError:	0.0373317487537861		Loss:	0.0019676744039836		Epoch:	51		MeanAbsoluteError:	0.0373317487537861
MeanAbsoluteError:	0.0375785119831562		Loss:	0.0025280290782186		Epoch:	54		MeanAbsoluteError:	0.0375785119831562
MeanAbsoluteError:	0.0352179519832134		Loss:	0.0023918469336682		Epoch:	57		MeanAbsoluteError:	0.0352179519832134
MeanAbsoluteError:	0.0242716856300831		Loss:	0.0011808074279023		Epoch:	60		MeanAbsoluteError:	0.0242716856300831
MeanAbsoluteError:	0.0271720122545958		Loss:	0.0013234067591839		Epoch:	63		MeanAbsoluteError:	0.0271720122545958
MeanAbsoluteError:	0.0299981348216534		Loss:	0.0016848013505556		Epoch:	66		MeanAbsoluteError:	0.0299981348216534
MeanAbsoluteError:	0.0241949055343866		Loss:	0.0011033957416657		Epoch:	69		MeanAbsoluteError:	0.0241949055343866
MeanAbsoluteError:	0.0474353320896626		Loss:	0.0040404630459047		Epoch:	72		MeanAbsoluteError:	0.0474353320896626
MeanAbsoluteError:	0.0351134352385998		Loss:	0.0024702129220324		Epoch:	75		MeanAbsoluteError:	0.0351134352385998
MeanAbsoluteError:	0.0298867318779230		Loss:	0.0014961652923375		Epoch:	78		MeanAbsoluteError:	0.0298867318779230
MeanAbsoluteError:	0.0298634022474289		Loss:	0.0018500328379949		Epoch:	81		MeanAbsoluteError:	0.0298634022474289
MeanAbsoluteError:	0.0293430909514427		Loss:	0.0014140093200175		Epoch:	84		MeanAbsoluteError:	0.0293430909514427
MeanAbsoluteError:	0.0286836437880993		Loss:	0.0015858735090920		Epoch:	87		MeanAbsoluteError:	0.0286836437880993
MeanAbsoluteError:	0.0326725468039513		Loss:	0.0023983207231920		Epoch:	90		MeanAbsoluteError:	0.0326725468039513
MeanAbsoluteError:	0.0307757258415222		Loss:	0.0015077245521492		Epoch:	93		MeanAbsoluteError:	0.0307757258415222

MeanAbsoluteError:	0.0472222045063972		Loss:	0.0035040079383180		Epoch:	19		MeanAbsoluteError:
MeanAbsoluteError:	0.0422460734844208		Loss:	0.0032011981820688		Epoch:	22		MeanAbsoluteError:
MeanAbsoluteError:	0.0365778394043446		Loss:	0.0023913827026263		Epoch:	25		MeanAbsoluteError:
MeanAbsoluteError:	0.0469057857990265		Loss:	0.0036701258671071		Epoch:	28		MeanAbsoluteError:
MeanAbsoluteError:	0.0323068872094154		Loss:	0.0021892809309065		Epoch:	31		MeanAbsoluteError:
MeanAbsoluteError:	0.0389392562210560		Loss:	0.0032040243136830		Epoch:	34		MeanAbsoluteError:
MeanAbsoluteError:	0.0333974249660969		Loss:	0.0018403186612496		Epoch:	37		MeanAbsoluteError:
MeanAbsoluteError:	0.0336916260421276		Loss:	0.0017928492036715		Epoch:	40		MeanAbsoluteError:
MeanAbsoluteError:	0.0365883186459541		Loss:	0.0027109532799971		Epoch:	43		MeanAbsoluteError:
MeanAbsoluteError:	0.0335726104676723		Loss:	0.0019785914461993		Epoch:	46		MeanAbsoluteError:
MeanAbsoluteError:	0.0375863574445248		Loss:	0.0024532506441964		Epoch:	49		MeanAbsoluteError:
MeanAbsoluteError:	0.0374349430203438		Loss:	0.0024859190203383		Epoch:	52		MeanAbsoluteError:
MeanAbsoluteError:	0.0324752889573574		Loss:	0.0016620607763928		Epoch:	55		MeanAbsoluteError:
MeanAbsoluteError:	0.0309160612523556		Loss:	0.0017615703815993		Epoch:	58		MeanAbsoluteError:
MeanAbsoluteError:	0.0346726626157761		Loss:	0.0018891181618008		Epoch:	61		MeanAbsoluteError:
MeanAbsoluteError:	0.0303509850054979		Loss:	0.0017382283528735		Epoch:	64		MeanAbsoluteError:
MeanAbsoluteError:	0.0351510681211948		Loss:	0.0018150453688577		Epoch:	67		MeanAbsoluteError:
MeanAbsoluteError:	0.0298524945974350		Loss:	0.0013440548458935		Epoch:	70		MeanAbsoluteError:
MeanAbsoluteError:	0.0350923351943493		Loss:	0.0018883133556561		Epoch:	73		MeanAbsoluteError:

MeanAbsoluteError:	0.0309339389204979		Loss:	0.0016711361573211		Epoch:	76		MeanAbsoluteError:	0.0309339389204979
MeanAbsoluteError:	0.0301052909344435		Loss:	0.0020206653529645		Epoch:	79		MeanAbsoluteError:	0.0301052909344435
MeanAbsoluteError:	0.0371831692755222		Loss:	0.0031639948720112		Epoch:	82		MeanAbsoluteError:	0.0371831692755222
MeanAbsoluteError:	0.0344323255121708		Loss:	0.0030389068415388		Epoch:	85		MeanAbsoluteError:	0.0344323255121708
MeanAbsoluteError:	0.0345389619469643		Loss:	0.0021257858440679		Epoch:	88		MeanAbsoluteError:	0.0345389619469643
MeanAbsoluteError:	0.0293699521571398		Loss:	0.0014905960332336		Epoch:	91		MeanAbsoluteError:	0.0293699521571398
MeanAbsoluteError:	0.0249335449188948		Loss:	0.0010290151694790		Epoch:	94		MeanAbsoluteError:	0.0249335449188948
MeanAbsoluteError:	0.0359574630856514		Loss:	0.0022995505215866		Epoch:	97		MeanAbsoluteError:	0.0359574630856514
MeanAbsoluteError:	0.0283001158386469		Loss:	0.0013590673583427		Epoch:	100		MeanAbsoluteError:	0.0283001158386469
MeanAbsoluteError:	0.0308013912290335		Loss:	0.0020014100342191		Epoch:	103		MeanAbsoluteError:	0.0308013912290335
MeanAbsoluteError:	0.0317067392170429		Loss:	0.0018965913041029		Epoch:	106		MeanAbsoluteError:	0.0317067392170429
MeanAbsoluteError:	0.0288515426218510		Loss:	0.0015225710262062		Epoch:	109		MeanAbsoluteError:	0.0288515426218510
MeanAbsoluteError:	0.0325191169977188		Loss:	0.0020198942261881		Epoch:	112		MeanAbsoluteError:	0.0325191169977188
MeanAbsoluteError:	0.0353016592562199		Loss:	0.0020487186848186		Epoch:	115		MeanAbsoluteError:	0.0353016592562199
MeanAbsoluteError:	0.0350049622356892		Loss:	0.0020128895827968		Epoch:	118		MeanAbsoluteError:	0.0350049622356892
MeanAbsoluteError:	0.0340956747531891		Loss:	0.0037404226854311		Epoch:	121		MeanAbsoluteError:	0.0340956747531891
MeanAbsoluteError:	0.0325993970036507		Loss:	0.0019238847557322		Epoch:	124		MeanAbsoluteError:	0.0325993970036507
MeanAbsoluteError:	0.0337807647883892		Loss:	0.0018033595489604		Epoch:	127		MeanAbsoluteError:	0.0337807647883892
MeanAbsoluteError:	0.0271592047065496		Loss:	0.0014736341046435		Epoch:	130		MeanAbsoluteError:	0.0271592047065496

MeanAbsoluteError: 0.0263482239097357		Loss: 0.0011180064466316		Epoch: 133		MeanAbsoluteError: 0.0263482239097357
MeanAbsoluteError: 0.0325757302343845		Loss: 0.0017013427734907		Epoch: 136		MeanAbsoluteError: 0.0325757302343845
MeanAbsoluteError: 0.0295941736549139		Loss: 0.0016438969677048		Epoch: 139		MeanAbsoluteError: 0.0295941736549139
MeanAbsoluteError: 0.0259032957255840		Loss: 0.0010365914037850		Epoch: 142		MeanAbsoluteError: 0.0259032957255840
MeanAbsoluteError: 0.0304586105048656		Loss: 0.0017409019055776		Epoch: 145		MeanAbsoluteError: 0.0304586105048656
MeanAbsoluteError: 0.0315777026116848		Loss: 0.0017776187575821		Epoch: 148		MeanAbsoluteError: 0.0315777026116848
MeanAbsoluteError: 0.0325266905128956		Loss: 0.0019013672676270		Epoch: 151		MeanAbsoluteError: 0.0325266905128956
MeanAbsoluteError: 0.0326800383627415		Loss: 0.0020008388590733		Epoch: 154		MeanAbsoluteError: 0.0326800383627415
MeanAbsoluteError: 0.0350924059748650		Loss: 0.0017653392195436		Epoch: 157		MeanAbsoluteError: 0.0350924059748650
MeanAbsoluteError: 0.0276425126940012		Loss: 0.0014459939473974		Epoch: 160		MeanAbsoluteError: 0.0276425126940012
MeanAbsoluteError: 0.0261540599167347		Loss: 0.0010607012622391		Epoch: 163		MeanAbsoluteError: 0.0261540599167347
MeanAbsoluteError: 0.0275248549878597		Loss: 0.0013768956081809		Epoch: 166		MeanAbsoluteError: 0.0275248549878597
MeanAbsoluteError: 0.0337137691676617		Loss: 0.0022094924851055		Epoch: 169		MeanAbsoluteError: 0.0337137691676617
MeanAbsoluteError: 0.0297052636742592		Loss: 0.0017340758432900		Epoch: 172		MeanAbsoluteError: 0.0297052636742592
MeanAbsoluteError: 0.0264049787074327		Loss: 0.0012700333359784		Epoch: 175		MeanAbsoluteError: 0.0264049787074327
MeanAbsoluteError: 0.0302285533398390		Loss: 0.0015202668769884		Epoch: 178		MeanAbsoluteError: 0.0302285533398390
MeanAbsoluteError: 0.0297722034156322		Loss: 0.0018084574257955		Epoch: 181		MeanAbsoluteError: 0.0297722034156322
MeanAbsoluteError: 0.0305019244551659		Loss: 0.0015438529933038		Epoch: 184		MeanAbsoluteError: 0.0305019244551659
MeanAbsoluteError: 0.0332957729697227		Loss: 0.0021334026241675		Epoch: 187		MeanAbsoluteError: 0.0332957729697227

MeanAbsoluteError: 0.0304834861308336 | Loss: 0.0015276887321046 | Epoch: 190 | MeanAbsoluteError: 0.0296339560300112 | Loss: 0.0014917478297970 | Epoch: 193 | MeanAbsoluteError: 0.0296960249543190 | Loss: 0.0017263874760829 | Epoch: 196 | MeanAbsoluteError: 0.0284142736345530 | Loss: 0.0016998049520355 | Epoch: 199 | MeanAbsoluteError: 0.0348243266344070 | Loss: 0.0021034266267504 | Epoch: 202 | MeanAbsoluteError: 0.0369440652430058 | Loss: 0.0027670118392312 | Epoch: 205 | MeanAbsoluteError: 0.0284225065261126 | Loss: 0.0012729114802953 | Epoch: 208 | MeanAbsoluteError: 0.0303502175956964 | Loss: 0.0015949354606814 | Epoch: 211 | MeanAbsoluteError: 0.0338517688214779 | Loss: 0.0020704973638723 | Epoch: 214 | MeanAbsoluteError: 0.0314059294760227 | Loss: 0.0020140493288636 | Early stopping at epoch 214

```
metric_name = type(fun_control["metric_torch"]).__name__
print(f"loss: {df_eval}, Cross-validated {metric_name}: {df_metrics}")
```

loss: 0.0017949270042923412, Cross-validated MeanAbsoluteError: 0.03246738761663437

19.10.4 Detailed Hyperparameter Plots

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

epochs: 17.99662907660928
optimizer: 0.13592593472751693
sgd_momentum: 100.0

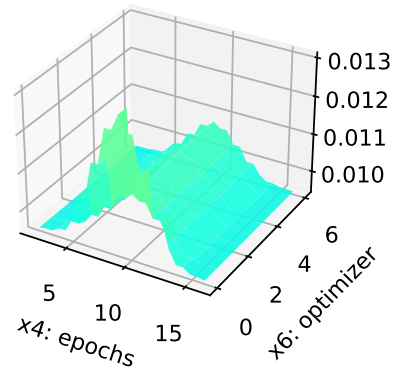
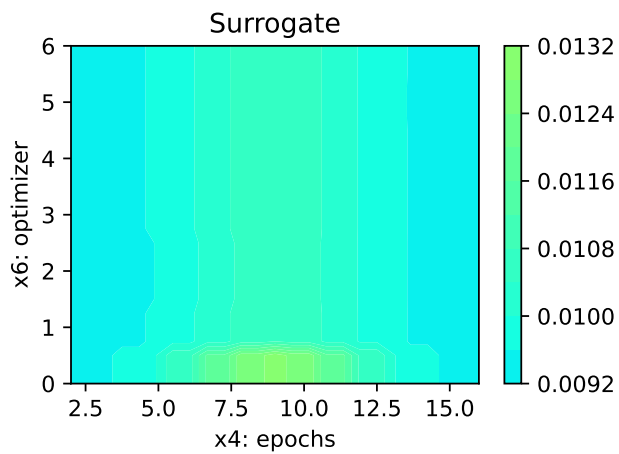
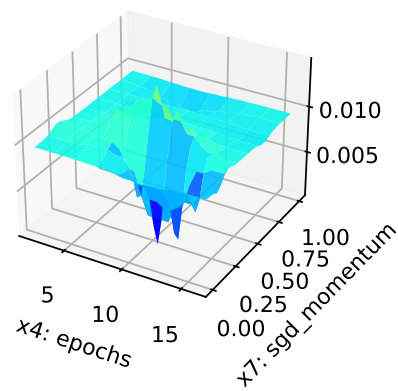
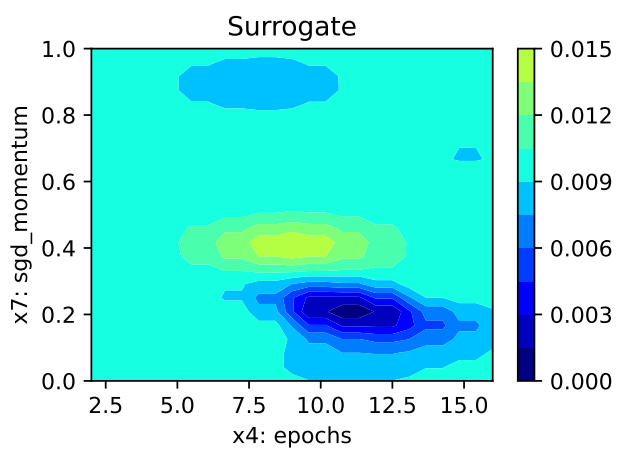
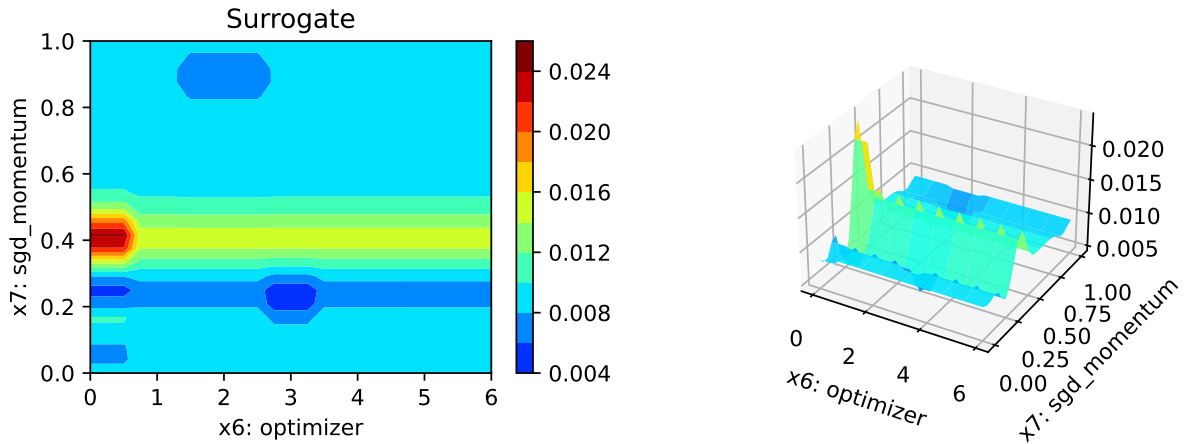


Figure 19.3: Contour plots.





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]"
```

spotPython	0.2.50
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
```

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_2023-06-28_04-57-06

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: 0.2142857164144516 | Loss: 2.3975850854601179 | Acc: 0.1367924528301887.

Epoch: 2 |

MAPK: 0.2239583432674408 | Loss: 2.3975568839481900 | Acc: 0.1603773584905660.

Epoch: 3 |

MAPK: 0.2135416567325592 | Loss: 2.3974567311150685 | Acc: 0.1367924528301887.

Epoch: 4 |

MAPK: 0.2075892835855484 | Loss: 2.3974663189479282 | Acc: 0.1273584905660377.

Epoch: 5 |

MAPK: 0.2120535820722580 | Loss: 2.3974057435989380 | Acc: 0.1415094339622641.
Epoch: 6 |

MAPK: 0.2046131044626236 | Loss: 2.3973787512098039 | Acc: 0.1367924528301887.
Epoch: 7 |

MAPK: 0.2053571492433548 | Loss: 2.3973322766167775 | Acc: 0.1132075471698113.
Epoch: 8 |

MAPK: 0.2165178805589676 | Loss: 2.3973527465547835 | Acc: 0.1367924528301887.
Returned to Spot: Validation loss: 2.3973527465547835

config: {'_L0': 6112, 'l1': 256, 'dropout_prob': 0.19379790035512987, 'lr_mult': 0.001, 'bat
Epoch: 1 |

MAPK: 0.2098765522241592 | Loss: 2.3962320486704507 | Acc: 0.1320754716981132.
Epoch: 2 |

MAPK: 0.2129629701375961 | Loss: 2.3962117742609093 | Acc: 0.1320754716981132.
Epoch: 3 |

MAPK: 0.2106481492519379 | Loss: 2.3962160216437445 | Acc: 0.1320754716981132.
Epoch: 4 |

MAPK: 0.2137345671653748 | Loss: 2.3961875791902894 | Acc: 0.1320754716981132.
Returned to Spot: Validation loss: 2.3961875791902894

config: {'_L0': 6112, 'l1': 4096, 'dropout_prob': 0.6759063718076167, 'lr_mult': 0.001, 'bat
Epoch: 1 |

MAPK: 0.1611635237932205 | Loss: 2.3978382034121819 | Acc: 0.1037735849056604.
Epoch: 2 |

MAPK: 0.1926100552082062 | Loss: 2.3974944892919288 | Acc: 0.1037735849056604.
Epoch: 3 |

MAPK: 0.2051886618137360 | Loss: 2.3973083383632154 | Acc: 0.1132075471698113.
Epoch: 4 |

MAPK: 0.2106917947530746 | Loss: 2.3973379877378358 | Acc: 0.1320754716981132.
Epoch: 5 |

MAPK: 0.2460691332817078 | Loss: 2.3968618388445870 | Acc: 0.1698113207547170.
Epoch: 6 |

MAPK: 0.2649371027946472 | Loss: 2.3964780006768569 | Acc: 0.1792452830188679.
Epoch: 7 |

MAPK: 0.2727987170219421 | Loss: 2.3963491916656494 | Acc: 0.1745283018867924.
Epoch: 8 |

MAPK: 0.2735848426818848 | Loss: 2.3956982392185138 | Acc: 0.1792452830188679.
Returned to Spot: Validation loss: 2.3956982392185138

config: {'_L0': 6112, 'l1': 128, 'dropout_prob': 0.37306669346546995, 'lr_mult': 0.001, 'batch_size': 128}
Epoch: 1 |

MAPK: 0.1533019095659256 | Loss: 2.3983729740358748 | Acc: 0.0849056603773585.
Epoch: 2 |

MAPK: 0.1470125913619995 | Loss: 2.3983781112814850 | Acc: 0.0849056603773585.
Epoch: 3 |

MAPK: 0.1556603759527206 | Loss: 2.3983388082036434 | Acc: 0.0849056603773585.
Epoch: 4 |

MAPK: 0.1533018797636032 | Loss: 2.3983615749287157 | Acc: 0.0849056603773585.
Returned to Spot: Validation loss: 2.3983615749287157

config: {'_L0': 6112, 'l1': 1024, 'dropout_prob': 0.870137281216666, 'lr_mult': 0.001, 'batch_size': 128}
Epoch: 1 |

MAPK: 0.1782407611608505 | Loss: 2.3979750032778138 | Acc: 0.1132075471698113.
Epoch: 2 |

MAPK: 0.1628086715936661 | Loss: 2.3979492982228598 | Acc: 0.0896226415094340.
Epoch: 3 |

MAPK: 0.1628086268901825 | Loss: 2.3982627568421542 | Acc: 0.0990566037735849.
Epoch: 4 |

MAPK: 0.1759259253740311 | Loss: 2.3979846636454263 | Acc: 0.1132075471698113.
Epoch: 5 |

MAPK: 0.1736111044883728 | Loss: 2.3979797186674894 | Acc: 0.0990566037735849.
Epoch: 6 |

MAPK: 0.1712962985038757 | Loss: 2.3979559827733925 | Acc: 0.0896226415094340.
Early stopping at epoch 5
Returned to Spot: Validation loss: 2.3979559827733925

config: {'_L0': 6112, 'l1': 128, 'dropout_prob': 0.1936430320861227, 'lr_mult': 0.001, 'batch
Epoch: 1 |

MAPK: 0.1574074029922485 | Loss: 2.3972890906863742 | Acc: 0.0613207547169811.
Epoch: 2 |

MAPK: 0.1574074029922485 | Loss: 2.3972662996362755 | Acc: 0.0613207547169811.
Epoch: 3 |

MAPK: 0.1527777761220932 | Loss: 2.3972931173112659 | Acc: 0.0613207547169811.
Epoch: 4 |

MAPK: 0.1589506268501282 | Loss: 2.3973015944163003 | Acc: 0.0613207547169811.
Epoch: 5 |

MAPK: 0.1574074029922485 | Loss: 2.3971795947463423 | Acc: 0.0613207547169811.
Epoch: 6 |

MAPK: 0.1558642238378525 | Loss: 2.3972731784537986 | Acc: 0.0613207547169811.
Epoch: 7 |

MAPK: 0.1574074029922485 | Loss: 2.3972059090932212 | Acc: 0.0613207547169811.
Epoch: 8 |

MAPK: 0.1543209999799728 | Loss: 2.3972016352194325 | Acc: 0.0613207547169811.
Returned to Spot: Validation loss: 2.3972016352194325
spotPython tuning: 2.3956982392185138 [##-----] 21.43%


```
MAPK: 0.1776729822158813 | Loss: 2.3977187084701828 | Acc: 0.1037735849056604.  
Epoch: 4 |
```

```
MAPK: 0.1705974787473679 | Loss: 2.3973627202915697 | Acc: 0.0990566037735849.  
Epoch: 5 |
```

```
MAPK: 0.1761006265878677 | Loss: 2.3973178368694379 | Acc: 0.0990566037735849.  
Epoch: 6 |
```

```
MAPK: 0.1761006414890289 | Loss: 2.3968935642602309 | Acc: 0.0990566037735849.  
Epoch: 7 |
```

```
MAPK: 0.1627358645200729 | Loss: 2.3967435697339616 | Acc: 0.0990566037735849.  
Epoch: 8 |
```

```
MAPK: 0.1682389825582504 | Loss: 2.3959970631689393 | Acc: 0.0990566037735849.  
Returned to Spot: Validation loss: 2.3959970631689393  
spotPython tuning: 2.3956982392185138 [#####] 100.00% Done...
```

```
<spotPython.spot.spot.Spot at 0x188c33e50>
```

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")
```

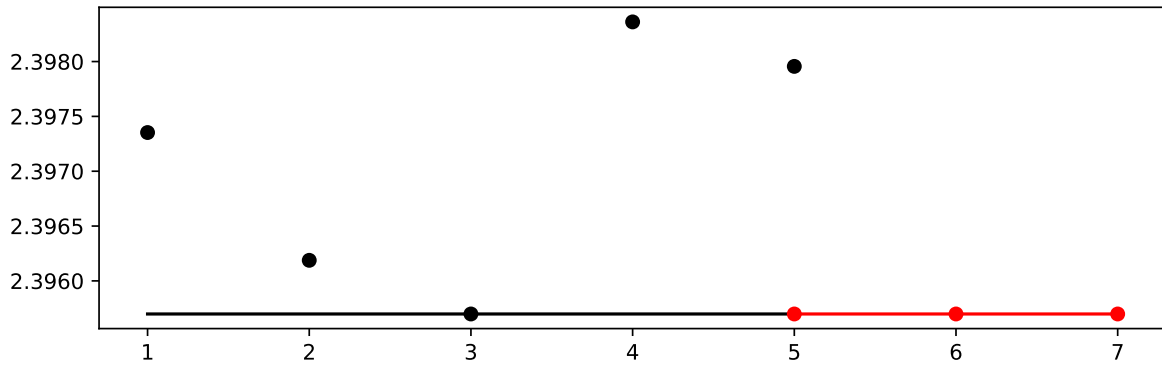


Figure 20.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

```
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	0.6759063718076167	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")
```

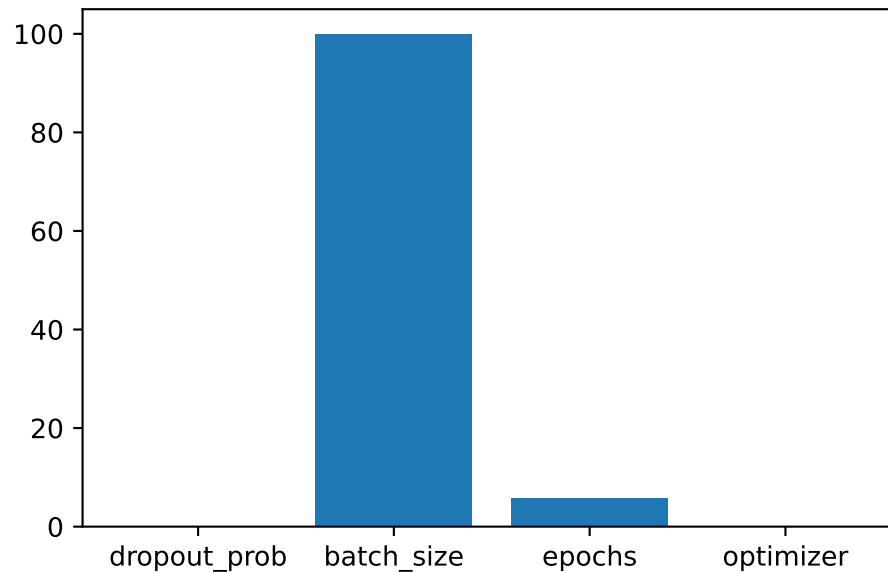



Figure 20.2: Variable importance plot, threshold 0.025.

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=0.6759063718076167, inplace=False)
  (dropout2): Dropout(p=0.33795318590380835, 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: 0.2106918096542358 | Loss: 2.3974566864517501 | Acc: 0.1084905660377359.

Epoch: 2 |

MAPK: 0.2028301656246185 | Loss: 2.3974239961156307 | Acc: 0.1084905660377359.

Epoch: 3 |

MAPK: 0.2240565568208694 | Loss: 2.3970022673876779 | Acc: 0.1367924528301887.

Epoch: 4 |

MAPK: 0.2122641652822495 | Loss: 2.3968893536981546 | Acc: 0.1132075471698113.

Epoch: 5 |

MAPK: 0.1981131583452225 | Loss: 2.3966420304100469 | Acc: 0.1084905660377359.

Epoch: 6 |

MAPK: 0.2138364613056183 | Loss: 2.3961606857911595 | Acc: 0.1084905660377359.

Epoch: 7 |

MAPK: 0.2091194540262222 | Loss: 2.3954507067518414 | Acc: 0.1084905660377359.

Epoch: 8 |

MAPK: 0.2036163210868835 | Loss: 2.3949626616711885 | Acc: 0.1084905660377359.

Returned to Spot: Validation loss: 2.3949626616711885

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: 0.2181648015975952 | Loss: 2.3941416124279580 | Acc: 0.1186440677966102.
Final evaluation: Validation loss: 2.394141612427958
Final evaluation: Validation metric: 0.21816480159759521
-----
```

```
(2.394141612427958, nan, tensor(0.2182))
```

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: 0.2111581712961197 | Loss: 2.3974618790513378 | Acc: 0.1101694915254237.
 Epoch: 2 |

 MAPK: 0.2005649805068970 | Loss: 2.3971990205473821 | Acc: 0.1059322033898305.
 Epoch: 3 |

 MAPK: 0.2266948819160461 | Loss: 2.3966194088176147 | Acc: 0.1355932203389831.
 Epoch: 4 |

 MAPK: 0.2203389406204224 | Loss: 2.3958373089968146 | Acc: 0.1355932203389831.
 Epoch: 5 |

 MAPK: 0.2217514067888260 | Loss: 2.3948350942741006 | Acc: 0.1355932203389831.
 Epoch: 6 |

 MAPK: 0.2168078869581223 | Loss: 2.3933165841183421 | Acc: 0.1271186440677966.
 Epoch: 7 |

 MAPK: 0.2210451662540436 | Loss: 2.3913153349342995 | Acc: 0.1271186440677966.
 Epoch: 8 |

 MAPK: 0.2203389406204224 | Loss: 2.3889158317598245 | Acc: 0.1271186440677966.
 Fold: 2
 Epoch: 1 |

 MAPK: 0.2168078571557999 | Loss: 2.3973561282885276 | Acc: 0.1313559322033898.
 Epoch: 2 |

 MAPK: 0.2118643969297409 | Loss: 2.3970147795596364 | Acc: 0.1228813559322034.
 Epoch: 3 |

 MAPK: 0.2168078869581223 | Loss: 2.3965749215271512 | Acc: 0.1186440677966102.
 Epoch: 4 |

 MAPK: 0.2302259504795074 | Loss: 2.3961181276935641 | Acc: 0.1186440677966102.
 Epoch: 5 |

MAPK: 0.2344632446765900 | Loss: 2.3955134517055447 | Acc: 0.1186440677966102.
Epoch: 6 |

MAPK: 0.2189265340566635 | Loss: 2.3942164344302679 | Acc: 0.1186440677966102.
Epoch: 7 |

MAPK: 0.2182203084230423 | Loss: 2.3929938118336564 | Acc: 0.1186440677966102.
Epoch: 8 |

MAPK: 0.2168079018592834 | Loss: 2.3914948802883340 | Acc: 0.1186440677966102.
Fold: 3
Epoch: 1 |

MAPK: 0.2146892398595810 | Loss: 2.3978472079260875 | Acc: 0.1404255319148936.
Epoch: 2 |

MAPK: 0.2217513918876648 | Loss: 2.3973214363647721 | Acc: 0.1276595744680851.
Epoch: 3 |

MAPK: 0.1878530830144882 | Loss: 2.3968886783567527 | Acc: 0.0978723404255319.
Epoch: 4 |

MAPK: 0.1949152350425720 | Loss: 2.3962746757571982 | Acc: 0.1148936170212766.
Epoch: 5 |

MAPK: 0.1843220293521881 | Loss: 2.3953310837179926 | Acc: 0.1021276595744681.
Epoch: 6 |

MAPK: 0.1857344508171082 | Loss: 2.3937102940122958 | Acc: 0.1106382978723404.
Epoch: 7 |

MAPK: 0.1913841664791107 | Loss: 2.3920364036398420 | Acc: 0.1106382978723404.
Epoch: 8 |

MAPK: 0.1878530830144882 | Loss: 2.3900211483745251 | Acc: 0.1148936170212766.

```
metric_name = type(fun_control["metric_torch"]).__name__  
print(f"loss: {df_eval}, Cross-validated {metric_name}: {df_metrics}")
```

loss: 2.390143953474228, Cross-validated MAPK: 0.2083333134651184

20.10.4 Detailed Hyperparameter Plots

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

dropout_prob: 0.16567008687914087
batch_size: 100.0
epochs: 5.786708972476294
optimizer: 0.07819175787408056

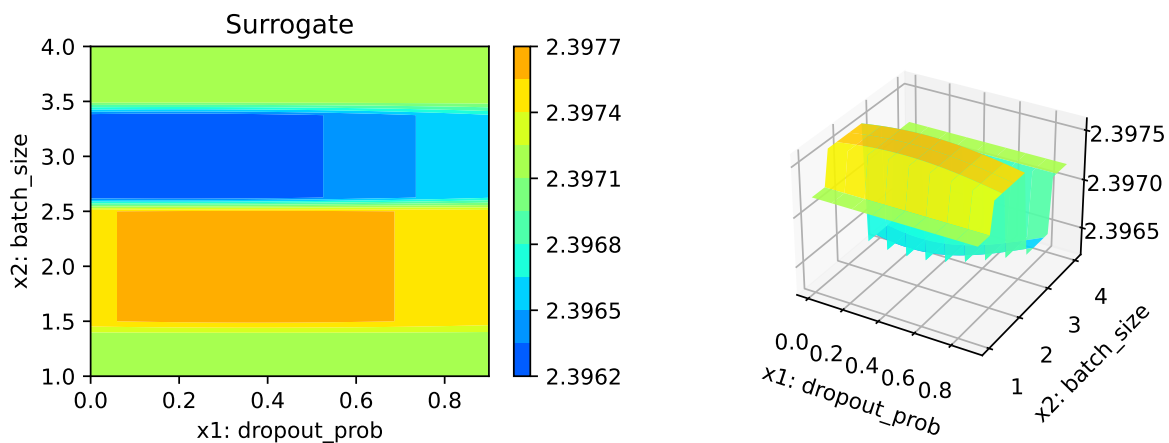
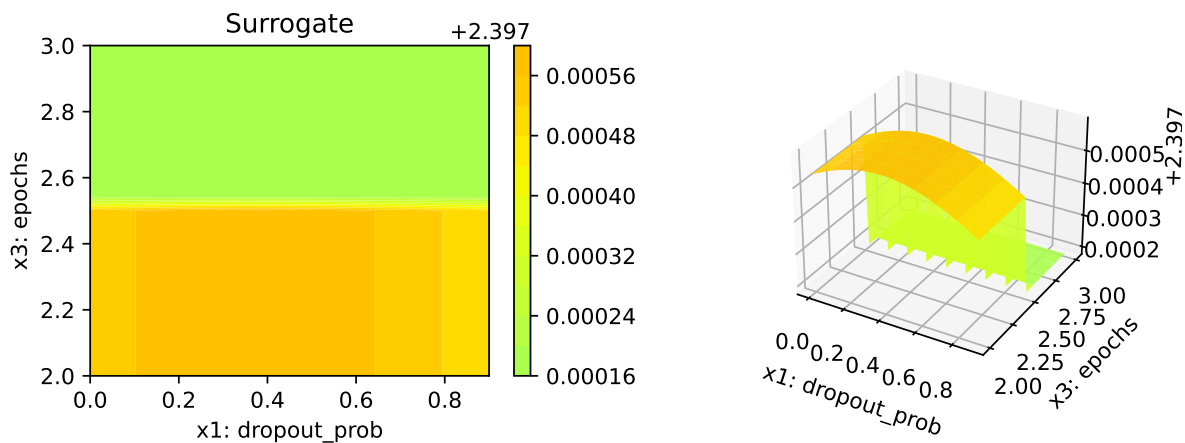
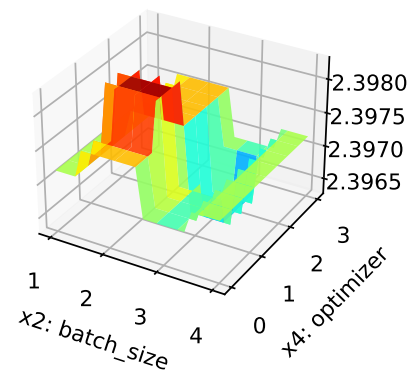
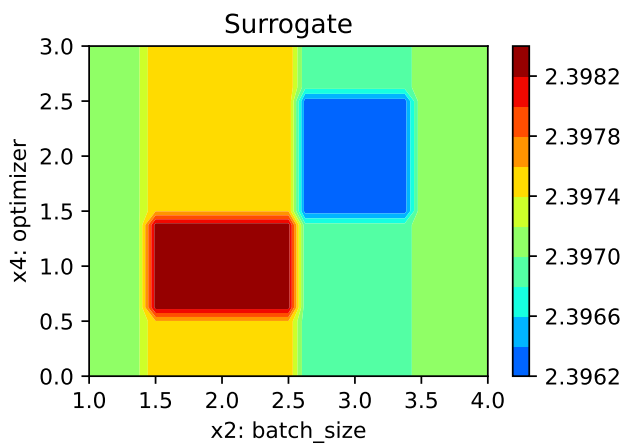
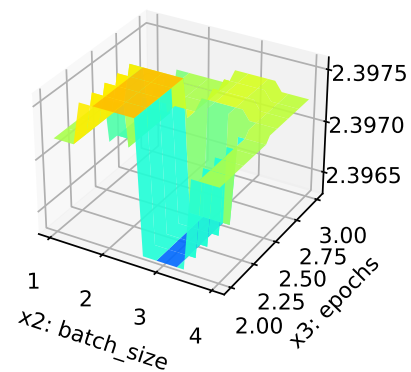
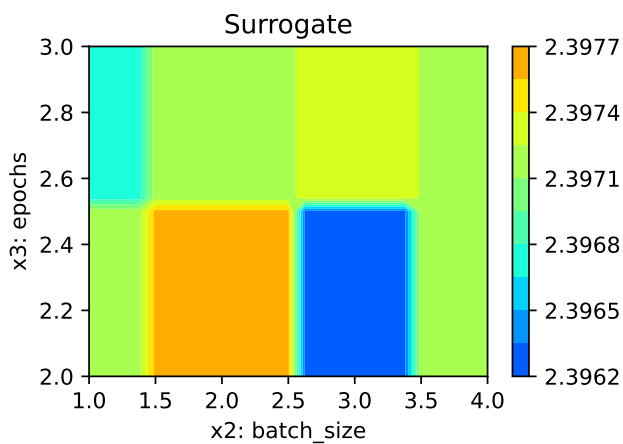
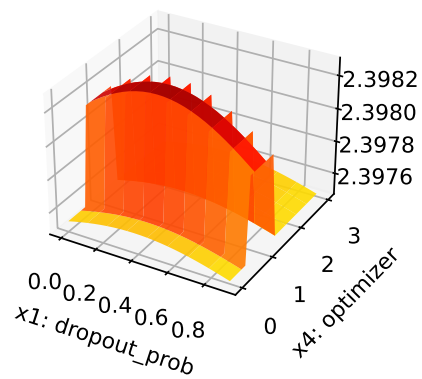
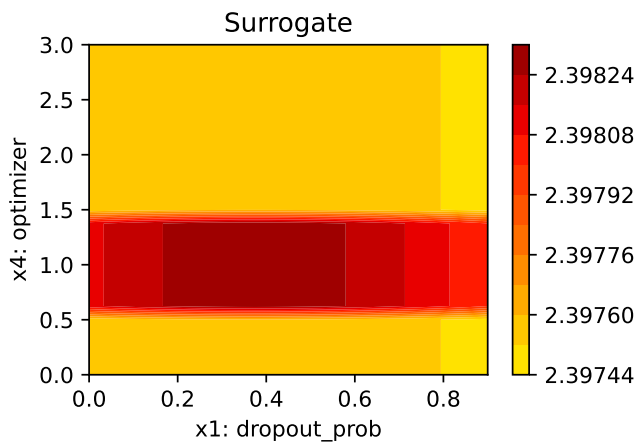
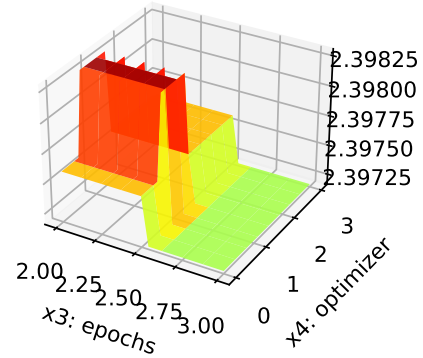
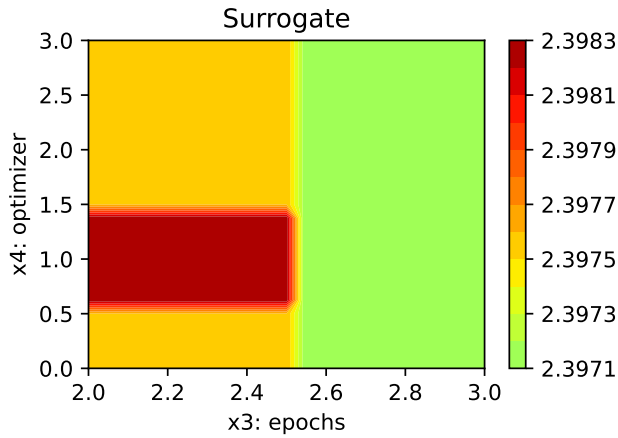


Figure 20.3: Contour plots.







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	0.2.50
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 = "auto" #"cpu" # "cuda:0"
WORKERS = 0
PREFIX="30"
```

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

`mps`

```
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(task="classification",
    tensorboard_path="./runs/" + experiment_name,
    num_workers=WORKERS,
    device=DEVICE)
```

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([[0., 0., 0., 0., 0., 1., 0., 1., 0., 0., 1., 1., 0., 1., 0., 1., 1., 0.,

```

1., 1., 1., 1., 1., 1., 1., 0., 1., 0., 1., 1., 0., 0., 0., 0., 0., 0.,
1., 0., 1., 0., 0., 0., 1., 0., 1., 1., 0., 1., 1., 0., 1., 1., 1.,
0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
[1., 1., 1., 1., 1., 0., 0., 1., 1., 0., 1., 1., 1., 0., 0., 1., 1., 0.,
1., 0., 0., 1., 1., 1., 1., 1., 1., 0., 1., 0., 0., 0., 0., 0., 1.,
0., 0., 0., 0., 0., 0., 1., 1., 0., 1., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
[0., 1., 0., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 1., 0., 1., 1., 0.,
0., 0., 1., 1., 0., 1., 1., 0., 1., 1., 1., 0., 1., 0., 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., 0., 0., 0.]]

```

Targets: `tensor([8, 4, 5])`

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': 0.1},
'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=[7,10])
fun_control = modify_hyper_parameter_bounds(fun_control, "epochs", bounds=[5,7])
fun_control = modify_hyper_parameter_bounds(fun_control, "batch_size", bounds=[3, 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': 7,
'upper': 10},
'epochs': {'type': 'int',
```



```

'default': 4,
'transform': 'transform_power_2_int',
'lower': 5,
'upper': 7},
'batch_size': {'type': 'int',
'default': 4,
'transform': 'transform_power_2_int',
'lower': 3,
'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': 0.1},
'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	7	10	transform_power_2_int
epochs	int	4	5	7	transform_power_2_int
batch_size	int	4	3	8	transform_power_2_int
act_fn	factor	ReLU	0	5	None
optimizer	factor	SGD	0	3	None
dropout_prob	float	0.01	0	0.1	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
```

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': 1024, 'epochs': 128, 'batch_size': 64, 'act_fn': ReLU(), 'optimizer': 'AdamW'}
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=1024, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.04375810986688453, inplace=False)
    (3): Linear(in_features=1024, out_features=512, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.04375810986688453, inplace=False)
    (6): Linear(in_features=512, out_features=512, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.04375810986688453, inplace=False)
    (9): Linear(in_features=512, out_features=256, bias=True)
    (10): ReLU()
  )
)

```

```

(11): Dropout(p=0.04375810986688453, inplace=False)
(12): Linear(in_features=256, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	2.241300106048584
val_acc	0.2968197762966156
val_loss	2.241300106048584
valid_mapk	0.39207175374031067

```

train_model result: {'valid_mapk': 0.39207175374031067, 'val_loss': 2.241300106048584, 'val_

```

```

config: {'l1': 128, 'epochs': 32, 'batch_size': 256, 'act_fn': LeakyReLU(), 'optimizer': 'Ad

```

```

model: NetLightBase(
  (act_fn): LeakyReLU()
  (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=0.005170658955305807, inplace=False)
    (3): Linear(in_features=128, out_features=64, bias=True)
    (4): LeakyReLU()
    (5): Dropout(p=0.005170658955305807, inplace=False)
    (6): Linear(in_features=64, out_features=64, bias=True)
    (7): LeakyReLU()
    (8): Dropout(p=0.005170658955305807, inplace=False)
    (9): Linear(in_features=64, out_features=32, bias=True)
    (10): LeakyReLU()
    (11): Dropout(p=0.005170658955305807, inplace=False)
    (12): Linear(in_features=32, out_features=11, bias=True)
  )
)
)

```

Validate metric	DataLoader 0
-----------------	--------------

hp_metric	2.30289363861084
val_acc	0.23674911260604858
val_loss	2.30289363861084
valid_mapk	0.30731576681137085

train_model result: {'valid_mapk': 0.30731576681137085, 'val_loss': 2.30289363861084, 'val_a

config: {'l1': 512, 'epochs': 64, 'batch_size': 16, 'act_fn': Swish(), 'optimizer': 'NAdam',

model: NetLightBase(

(act_fn): Swish()

(train_mapk): MAPK()

(valid_mapk): MAPK()

(test_mapk): MAPK()

(layers): Sequential(

(0): Linear(in_features=64, out_features=512, bias=True)

(1): Swish()

(2): Dropout(p=0.08834550718769361, inplace=False)

(3): Linear(in_features=512, out_features=256, bias=True)

(4): Swish()

(5): Dropout(p=0.08834550718769361, inplace=False)

(6): Linear(in_features=256, out_features=256, bias=True)

(7): Swish()

(8): Dropout(p=0.08834550718769361, inplace=False)

(9): Linear(in_features=256, out_features=128, bias=True)

(10): Swish()

(11): Dropout(p=0.08834550718769361, inplace=False)

(12): Linear(in_features=128, out_features=11, bias=True)

)

)

Validate metric

DataLoader 0

hp_metric	2.490037441253662
val_acc	0.05300353467464447
val_loss	2.490037441253662
valid_mapk	0.12957701086997986

train_model result: {'valid_mapk': 0.12957701086997986, 'val_loss': 2.490037441253662, 'val_a

```

config: {'l1': 256, 'epochs': 64, 'batch_size': 16, 'act_fn': Sigmoid(), 'optimizer': 'Adam'}
model: NetLightBase(
  (act_fn): Sigmoid()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=256, bias=True)
    (1): Sigmoid()
    (2): Dropout(p=0.07563714253500024, inplace=False)
    (3): Linear(in_features=256, out_features=128, bias=True)
    (4): Sigmoid()
    (5): Dropout(p=0.07563714253500024, inplace=False)
    (6): Linear(in_features=128, out_features=128, bias=True)
    (7): Sigmoid()
    (8): Dropout(p=0.07563714253500024, inplace=False)
    (9): Linear(in_features=128, out_features=64, bias=True)
    (10): Sigmoid()
    (11): Dropout(p=0.07563714253500024, inplace=False)
    (12): Linear(in_features=64, out_features=11, bias=True)
  )
)

```

Validate metric	DataLoader 0
hp_metric	2.237091064453125
val_acc	0.30035334825515747
val_loss	2.237091064453125
valid_mapk	0.38673195242881775

```

train_model result: {'valid_mapk': 0.38673195242881775, 'val_loss': 2.237091064453125, 'val_
config: {'l1': 256, 'epochs': 128, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax'}
model: NetLightBase(
  (act_fn): ReLU()
  (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=0.02833523179697884, inplace=False)
(3): Linear(in_features=256, out_features=128, bias=True)
(4): ReLU()
(5): Dropout(p=0.02833523179697884, inplace=False)
(6): Linear(in_features=128, out_features=128, bias=True)
(7): ReLU()
(8): Dropout(p=0.02833523179697884, inplace=False)
(9): Linear(in_features=128, out_features=64, bias=True)
(10): ReLU()
(11): Dropout(p=0.02833523179697884, inplace=False)
(12): Linear(in_features=64, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	2.314218759536743
val_acc	0.22968198359012604
val_loss	2.314218759536743
valid_mapk	0.2855902910232544

train_model result: {'valid_mapk': 0.2855902910232544, 'val_loss': 2.314218759536743, 'val_a

```

config: {'l1': 256, 'epochs': 32, 'batch_size': 32, 'act_fn': ELU(), 'optimizer': 'Adamax',
model: NetLightBase(
  (act_fn): ELU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=256, bias=True)
    (1): ELU()
    (2): Dropout(p=0.07366997122359899, inplace=False)
    (3): Linear(in_features=256, out_features=128, bias=True)
    (4): ELU()
    (5): Dropout(p=0.07366997122359899, inplace=False)
    (6): Linear(in_features=128, out_features=128, bias=True)
    (7): ELU()

```

```

(8): Dropout(p=0.07366997122359899, inplace=False)
(9): Linear(in_features=128, out_features=64, bias=True)
(10): ELU()
(11): Dropout(p=0.07366997122359899, inplace=False)
(12): Linear(in_features=64, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	2.2027337551116943
val_acc	0.3321554660797119
val_loss	2.2027337551116943
valid_mapk	0.4326346218585968

train_model result: {'valid_mapk': 0.4326346218585968, 'val_loss': 2.2027337551116943, 'val_

spotPython tuning: 2.2027337551116943 [#-----] 5.86%

config: {'l1': 512, 'epochs': 64, 'batch_size': 128, 'act_fn': ELU(), 'optimizer': 'AdamW',

```

model: NetLightBase(
  (act_fn): ELU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=512, bias=True)
    (1): ELU()
    (2): Dropout(p=0.06519059541620381, inplace=False)
    (3): Linear(in_features=512, out_features=256, bias=True)
    (4): ELU()
    (5): Dropout(p=0.06519059541620381, inplace=False)
    (6): Linear(in_features=256, out_features=256, bias=True)
    (7): ELU()
    (8): Dropout(p=0.06519059541620381, inplace=False)
    (9): Linear(in_features=256, out_features=128, bias=True)
    (10): ELU()
  )
)

```

```

(11): Dropout(p=0.06519059541620381, inplace=False)
(12): Linear(in_features=128, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	nan
val_acc	0.09540636092424393
val_loss	nan
valid_mapk	0.18410813808441162

train_model result: {'valid_mapk': 0.18410813808441162, 'val_loss': nan, 'val_acc': 0.09540636092424393}

spotPython tuning: 2.2027337551116943 [#-----] 9.87%

```

config: {'l1': 512, 'epochs': 64, 'batch_size': 128, 'act_fn': ELU(), 'optimizer': 'AdamW',
model: NetLightBase(
  (act_fn): ELU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=512, bias=True)
    (1): ELU()
    (2): Dropout(p=0.06519059541620381, inplace=False)
    (3): Linear(in_features=512, out_features=256, bias=True)
    (4): ELU()
    (5): Dropout(p=0.06519059541620381, inplace=False)
    (6): Linear(in_features=256, out_features=256, bias=True)
    (7): ELU()
    (8): Dropout(p=0.06519059541620381, inplace=False)
    (9): Linear(in_features=256, out_features=128, bias=True)
    (10): ELU()
    (11): Dropout(p=0.06519059541620381, inplace=False)
    (12): Linear(in_features=128, out_features=11, bias=True)
  )
)
)

```

Validate metric	DataLoader 0
hp_metric	2.344399929046631
val_acc	0.1978798657655716
val_loss	2.344399929046631
valid_mapk	0.3192997872829437

train_model result: {'valid_mapk': 0.3192997872829437, 'val_loss': 2.344399929046631, 'val_a

spotPython tuning: 2.2027337551116943 [#-----] 14.00%

```

config: {'l1': 256, 'epochs': 32, 'batch_size': 64, 'act_fn': ELU(), 'optimizer': 'Adamax',
model: NetLightBase(
  (act_fn): ELU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=256, bias=True)
    (1): ELU()
    (2): Dropout(p=0.07076625284388272, inplace=False)
    (3): Linear(in_features=256, out_features=128, bias=True)
    (4): ELU()
    (5): Dropout(p=0.07076625284388272, inplace=False)
    (6): Linear(in_features=128, out_features=128, bias=True)
    (7): ELU()
    (8): Dropout(p=0.07076625284388272, inplace=False)
    (9): Linear(in_features=128, out_features=64, bias=True)
    (10): ELU()
    (11): Dropout(p=0.07076625284388272, inplace=False)
    (12): Linear(in_features=64, out_features=11, bias=True)
  )
)

```

Validate metric	DataLoader 0
hp_metric	2.2551815509796143

val_acc	0.27915194630622864
val_loss	2.2551815509796143
valid_mapk	0.375964492559433

train_model result: {'valid_mapk': 0.375964492559433, 'val_loss': 2.2551815509796143, 'val_a

spotPython tuning: 2.2027337551116943 [##-----] 18.42%

config: {'l1': 256, 'epochs': 32, 'batch_size': 8, 'act_fn': ELU(), 'optimizer': 'Adamax', '
model: NetLightBase(
(act_fn): ELU()
(train_mapk): MAPK()
(valid_mapk): MAPK()
(test_mapk): MAPK()
(layers): Sequential(
(0): Linear(in_features=64, out_features=256, bias=True)
(1): ELU()
(2): Dropout(p=0.08421016393076294, inplace=False)
(3): Linear(in_features=256, out_features=128, bias=True)
(4): ELU()
(5): Dropout(p=0.08421016393076294, inplace=False)
(6): Linear(in_features=128, out_features=128, bias=True)
(7): ELU()
(8): Dropout(p=0.08421016393076294, inplace=False)
(9): Linear(in_features=128, out_features=64, bias=True)
(10): ELU()
(11): Dropout(p=0.08421016393076294, inplace=False)
(12): Linear(in_features=64, out_features=11, bias=True)
)
)

Validate metric	DataLoader 0
hp_metric	2.286494255065918
val_acc	0.24028268456459045
val_loss	2.286494255065918
valid_mapk	0.35300931334495544

train_model result: {'valid_mapk': 0.35300931334495544, 'val_loss': 2.286494255065918, 'val_acc': 0.2826855182647705}

spotPython tuning: 2.2027337551116943 [####-----] 35.29%

config: {'l1': 512, 'epochs': 64, 'batch_size': 64, 'act_fn': ReLU(), 'optimizer': 'Adamax', 'loss': 'mapk'}

```
model: NetLightBase(  
  (act_fn): ReLU()  
  (train_mapk): MAPK()  
  (valid_mapk): MAPK()  
  (test_mapk): MAPK()  
  (layers): Sequential(  
    (0): Linear(in_features=64, out_features=512, bias=True)  
    (1): ReLU()  
    (2): Dropout(p=0.062434519773562756, inplace=False)  
    (3): Linear(in_features=512, out_features=256, bias=True)  
    (4): ReLU()  
    (5): Dropout(p=0.062434519773562756, inplace=False)  
    (6): Linear(in_features=256, out_features=256, bias=True)  
    (7): ReLU()  
    (8): Dropout(p=0.062434519773562756, inplace=False)  
    (9): Linear(in_features=256, out_features=128, bias=True)  
    (10): ReLU()  
    (11): Dropout(p=0.062434519773562756, inplace=False)  
    (12): Linear(in_features=128, out_features=11, bias=True)  
  )  
)
```

Validate metric	DataLoader 0
hp_metric	2.25866961479187
val_acc	0.2826855182647705
val_loss	2.25866961479187
valid_mapk	0.3631751835346222

train_model result: {'valid_mapk': 0.3631751835346222, 'val_loss': 2.25866961479187, 'val_acc': 0.2826855182647705}

spotPython tuning: 2.2027337551116943 [####-----] 44.09%

```
config: {'l1': 128, 'epochs': 32, 'batch_size': 32, 'act_fn': LeakyReLU(), 'optimizer': 'Adam
```

```
model: NetLightBase(  
  (act_fn): LeakyReLU()  
  (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=0.1, inplace=False)  
    (3): Linear(in_features=128, out_features=64, bias=True)  
    (4): LeakyReLU()  
    (5): Dropout(p=0.1, inplace=False)  
    (6): Linear(in_features=64, out_features=64, bias=True)  
    (7): LeakyReLU()  
    (8): Dropout(p=0.1, inplace=False)  
    (9): Linear(in_features=64, out_features=32, bias=True)  
    (10): LeakyReLU()  
    (11): Dropout(p=0.1, inplace=False)  
    (12): Linear(in_features=32, out_features=11, bias=True)  
  )  
)
```

Validate metric	DataLoader 0
hp_metric	2.2462005615234375
val_acc	0.28975266218185425
val_loss	2.2462005615234375
valid_mapk	0.38983193039894104

```
train_model result: {'valid_mapk': 0.38983193039894104, 'val_loss': 2.2462005615234375, 'val
```

```
spotPython tuning: 2.2027337551116943 [####-----] 51.89%
```

```
config: {'l1': 512, 'epochs': 32, 'batch_size': 32, 'act_fn': LeakyReLU(), 'optimizer': 'Adam
```

```

model: NetLightBase(
  (act_fn): LeakyReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=512, bias=True)
    (1): LeakyReLU()
    (2): Dropout(p=0.1, inplace=False)
    (3): Linear(in_features=512, out_features=256, bias=True)
    (4): LeakyReLU()
    (5): Dropout(p=0.1, inplace=False)
    (6): Linear(in_features=256, out_features=256, bias=True)
    (7): LeakyReLU()
    (8): Dropout(p=0.1, inplace=False)
    (9): Linear(in_features=256, out_features=128, bias=True)
    (10): LeakyReLU()
    (11): Dropout(p=0.1, inplace=False)
    (12): Linear(in_features=128, out_features=11, bias=True)
  )
)

```

Validate metric	DataLoader 0
hp_metric	2.295387029647827
val_acc	0.25088340044021606
val_loss	2.295387029647827
valid_mapk	0.33101850748062134

train_model result: {'valid_mapk': 0.33101850748062134, 'val_loss': 2.295387029647827, 'val_

spotPython tuning: 2.2027337551116943 [#####----] 61.53%

```

config: {'l1': 256, 'epochs': 32, 'batch_size': 32, 'act_fn': ELU(), 'optimizer': 'Adamax',
model: NetLightBase(
  (act_fn): ELU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()

```



```

(test_mapk): MAPK()
(layers): Sequential(
  (0): Linear(in_features=64, out_features=256, bias=True)
  (1): ELU()
  (2): Dropout(p=0.08452955327295905, inplace=False)
  (3): Linear(in_features=256, out_features=128, bias=True)
  (4): ELU()
  (5): Dropout(p=0.08452955327295905, inplace=False)
  (6): Linear(in_features=128, out_features=128, bias=True)
  (7): ELU()
  (8): Dropout(p=0.08452955327295905, inplace=False)
  (9): Linear(in_features=128, out_features=64, bias=True)
  (10): ELU()
  (11): Dropout(p=0.08452955327295905, inplace=False)
  (12): Linear(in_features=64, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	2.2701313495635986
val_acc	0.2720848023891449
val_loss	2.2701313495635986
valid_mapk	0.33466222882270813

train_model result: {'valid_mapk': 0.33466222882270813, 'val_loss': 2.2701313495635986, 'val.

spotPython tuning: 2.2027337551116943 [#####---] 67.94%

```

config: {'l1': 128, 'epochs': 64, 'batch_size': 32, 'act_fn': LeakyReLU(), 'optimizer': 'Ada
model: NetLightBase(
  (act_fn): LeakyReLU()
  (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=0.025408754150260268, inplace=False)
(3): Linear(in_features=128, out_features=64, bias=True)
(4): LeakyReLU()
(5): Dropout(p=0.025408754150260268, inplace=False)
(6): Linear(in_features=64, out_features=64, bias=True)
(7): LeakyReLU()
(8): Dropout(p=0.025408754150260268, inplace=False)
(9): Linear(in_features=64, out_features=32, bias=True)
(10): LeakyReLU()
(11): Dropout(p=0.025408754150260268, inplace=False)
(12): Linear(in_features=32, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	2.2508442401885986
val_acc	0.27561837434768677
val_loss	2.2508442401885986
valid_mapk	0.39872682094573975

train_model result: {'valid_mapk': 0.39872682094573975, 'val_loss': 2.2508442401885986, 'val.

spotPython tuning: 2.2027337551116943 [#####---] 74.82%

config: {'l1': 128, 'epochs': 32, 'batch_size': 32, 'act_fn': Tanh(), 'optimizer': 'Adamax',

```

model: NetLightBase(
  (act_fn): Tanh()
  (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=0.08321943424467138, inplace=False)
    (3): Linear(in_features=128, out_features=64, bias=True)

```

```

(4): Tanh()
(5): Dropout(p=0.08321943424467138, inplace=False)
(6): Linear(in_features=64, out_features=64, bias=True)
(7): Tanh()
(8): Dropout(p=0.08321943424467138, inplace=False)
(9): Linear(in_features=64, out_features=32, bias=True)
(10): Tanh()
(11): Dropout(p=0.08321943424467138, inplace=False)
(12): Linear(in_features=32, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	2.254368305206299
val_acc	0.27561837434768677
val_loss	2.254368305206299
valid_mapk	0.39606910943984985

train_model result: {'valid_mapk': 0.39606910943984985, 'val_loss': 2.254368305206299, 'val_acc': 0.27561837434768677}

spotPython tuning: 2.2027337551116943 [#####--] 80.21%

config: {'l1': 256, 'epochs': 32, 'batch_size': 16, 'act_fn': ReLU(), 'optimizer': 'Adam', 'train_mapk': MAPK(), 'valid_mapk': MAPK(), 'test_mapk': MAPK()}

```

model: NetLightBase(
  (act_fn): ReLU()
  (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=0.047198681672901734, inplace=False)
    (3): Linear(in_features=256, out_features=128, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.047198681672901734, inplace=False)
  )
)

```

```

(6): Linear(in_features=128, out_features=128, bias=True)
(7): ReLU()
(8): Dropout(p=0.047198681672901734, inplace=False)
(9): Linear(in_features=128, out_features=64, bias=True)
(10): ReLU()
(11): Dropout(p=0.047198681672901734, inplace=False)
(12): Linear(in_features=64, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	2.2727861404418945
val_acc	0.2614840865135193
val_loss	2.2727861404418945
valid_mapk	0.3576388955116272

train_model result: {'valid_mapk': 0.3576388955116272, 'val_loss': 2.2727861404418945, 'val_

spotPython tuning: 2.2027337551116943 [#####-] 88.12%

config: {'l1': 256, 'epochs': 64, 'batch_size': 32, 'act_fn': LeakyReLU(), 'optimizer': 'Ada

```

model: NetLightBase(
  (act_fn): LeakyReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=256, bias=True)
    (1): LeakyReLU()
    (2): Dropout(p=0.06905041947245856, inplace=False)
    (3): Linear(in_features=256, out_features=128, bias=True)
    (4): LeakyReLU()
    (5): Dropout(p=0.06905041947245856, inplace=False)
    (6): Linear(in_features=128, out_features=128, bias=True)
    (7): LeakyReLU()
  )
)

```

```

(8): Dropout(p=0.06905041947245856, inplace=False)
(9): Linear(in_features=128, out_features=64, bias=True)
(10): LeakyReLU()
(11): Dropout(p=0.06905041947245856, inplace=False)
(12): Linear(in_features=64, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	2.2727742195129395
val_acc	0.26501765847206116
val_loss	2.2727742195129395
valid_mapk	0.36829131841659546

train_model result: {'valid_mapk': 0.36829131841659546, 'val_loss': 2.2727742195129395, 'val

spotPython tuning: 2.2027337551116943 [#####] 95.04%

config: {'l1': 1024, 'epochs': 64, 'batch_size': 64, 'act_fn': Tanh(), 'optimizer': 'AdamW',

```

model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=1024, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.05330594410414069, inplace=False)
    (3): Linear(in_features=1024, out_features=512, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.05330594410414069, inplace=False)
    (6): Linear(in_features=512, out_features=512, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.05330594410414069, inplace=False)
    (9): Linear(in_features=512, out_features=256, bias=True)
    (10): Tanh()
  )
)

```

```

(11): Dropout(p=0.05330594410414069, inplace=False)
(12): Linear(in_features=256, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	2.3239688873291016
val_acc	0.21554769575595856
val_loss	2.3239688873291016
valid_mapk	0.2965470552444458

```
train_model result: {'valid_mapk': 0.2965470552444458, 'val_loss': 2.3239688873291016, 'val_a
```

```
spotPython tuning: 2.2027337551116943 [#####] 100.00% Done...
```

```
<spotPython.spot.spot.Spot at 0x136e239d0>
```

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")

```

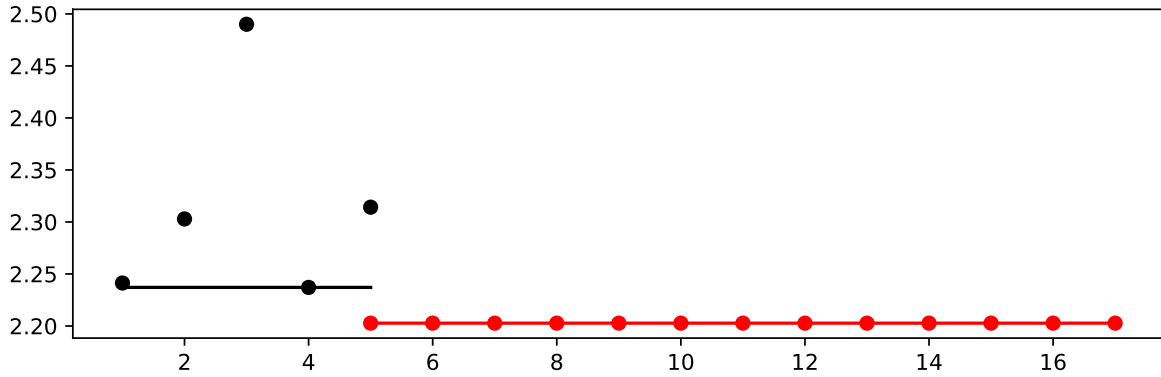


Figure 21.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

```
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
l1	int	3	7.0	10.0	8.0	transform_l
epochs	int	4	5.0	7.0	5.0	transform_l
batch_size	int	4	3.0	8.0	5.0	transform_l
act_fn	factor	ReLU	0.0	5.0	4.0	None
optimizer	factor	SGD	0.0	3.0	2.0	None
dropout_prob	float	0.01	0.0	0.1	0.07366997122359899	None
lr_mult	float	1.0	0.1	10.0	2.7294359664218484	None
patience	int	2	2.0	6.0	5.0	transform_l
initialization	factor	Default	0.0	2.0	1.0	None

```
spot_tuner.plot_importance(threshold=0.025,
    filename="./figures/" + experiment_name+"_importance.png")
```

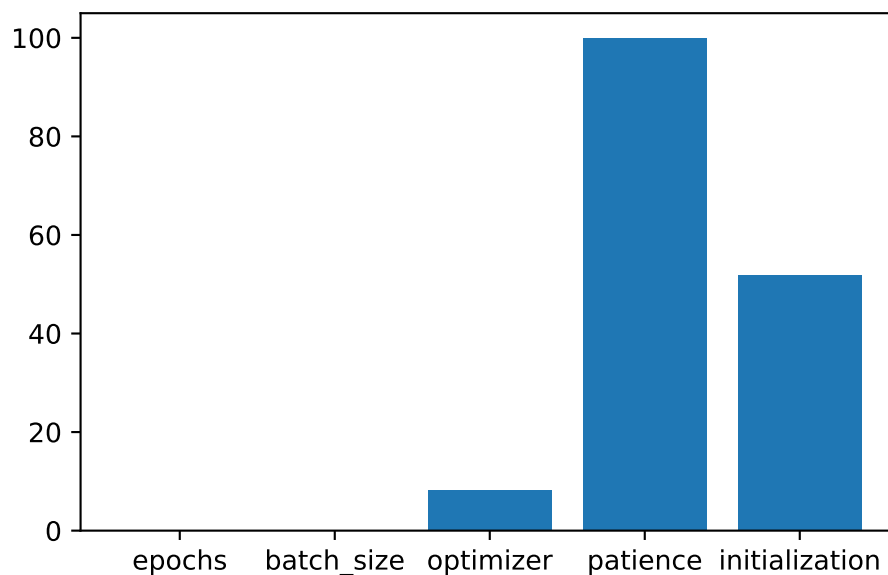


Figure 21.2: Variable importance plot, threshold 0.025.

21.10.1 Get the Tuned Architecture

```
from spotPython.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)
config
```

```
{'l1': 256,
 'epochs': 32,
 'batch_size': 32,
 'act_fn': ELU(),
 'optimizer': 'Adamax',
 'dropout_prob': 0.07366997122359899,
 'lr_mult': 2.7294359664218484,
 'patience': 32,
 'initialization': 'Kaiming'}
```

- Test on the full data set

```
from spotPython.light.traintest import test_model
test_model(config, fun_control)
```



```

model: NetLightBase(
  (act_fn): ELU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=256, bias=True)
    (1): ELU()
    (2): Dropout(p=0.07366997122359899, inplace=False)
    (3): Linear(in_features=256, out_features=128, bias=True)
    (4): ELU()
    (5): Dropout(p=0.07366997122359899, inplace=False)
    (6): Linear(in_features=128, out_features=128, bias=True)
    (7): ELU()
    (8): Dropout(p=0.07366997122359899, inplace=False)
    (9): Linear(in_features=128, out_features=64, bias=True)
    (10): ELU()
    (11): Dropout(p=0.07366997122359899, inplace=False)
    (12): Linear(in_features=64, out_features=11, bias=True)
  )
)

```

Test metric	DataLoader 0
test_mapk_epoch	0.5645380616188049
val_acc	0.5275813341140747
val_loss	2.0135626792907715

test_model result: {'test_mapk_epoch': 0.5645380616188049, 'val_loss': 2.0135626792907715, 'val_acc': 0.5275813341140747}

(2.0135626792907715, 0.5275813341140747)

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(
  (act_fn): ELU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=256, bias=True)
    (1): ELU()
    (2): Dropout(p=0.07366997122359899, inplace=False)
    (3): Linear(in_features=256, out_features=128, bias=True)
    (4): ELU()
    (5): Dropout(p=0.07366997122359899, inplace=False)
    (6): Linear(in_features=128, out_features=128, bias=True)
    (7): ELU()
    (8): Dropout(p=0.07366997122359899, inplace=False)
    (9): Linear(in_features=128, out_features=64, bias=True)
    (10): ELU()
    (11): Dropout(p=0.07366997122359899, inplace=False)
    (12): Linear(in_features=64, out_features=11, bias=True)
  )
)
k: 0
model: NetLightBase(
  (act_fn): ELU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=256, bias=True)
    (1): ELU()
    (2): Dropout(p=0.07366997122359899, inplace=False)
    (3): Linear(in_features=256, out_features=128, bias=True)
    (4): ELU()
    (5): Dropout(p=0.07366997122359899, inplace=False)
    (6): Linear(in_features=128, out_features=128, bias=True)
    (7): ELU()
    (8): Dropout(p=0.07366997122359899, inplace=False)
```

```

(9): Linear(in_features=128, out_features=64, bias=True)
(10): ELU()
(11): Dropout(p=0.07366997122359899, inplace=False)
(12): Linear(in_features=64, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	2.24653959274292
val_acc	0.2957746386528015
val_loss	2.24653959274292
valid_mapk	0.3958333432674408

train_model result: {'valid_mapk': 0.3958333432674408, 'val_loss': 2.24653959274292, 'val_acc': 0.2957746386528015, 'valid_mapk': 0.3958333432674408}

```

model: NetLightBase(
  (act_fn): ELU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=256, bias=True)
    (1): ELU()
    (2): Dropout(p=0.07366997122359899, inplace=False)
    (3): Linear(in_features=256, out_features=128, bias=True)
    (4): ELU()
    (5): Dropout(p=0.07366997122359899, inplace=False)
    (6): Linear(in_features=128, out_features=128, bias=True)
    (7): ELU()
    (8): Dropout(p=0.07366997122359899, inplace=False)
    (9): Linear(in_features=128, out_features=64, bias=True)
    (10): ELU()
    (11): Dropout(p=0.07366997122359899, inplace=False)
    (12): Linear(in_features=64, out_features=11, bias=True)
  )
)
)

```

Validate metric	DataLoader 0
-----------------	--------------

hp_metric	2.088435649871826
val_acc	0.43661972880363464
val_loss	2.088435649871826
valid_mapk	0.5994543433189392

train_model result: {'valid_mapk': 0.5994543433189392, 'val_loss': 2.088435649871826, 'val_acc': 0.43661972880363464, 'valid_mapk': 0.5994543433189392}

```
model: NetLightBase(
  (act_fn): ELU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=256, bias=True)
    (1): ELU()
    (2): Dropout(p=0.07366997122359899, inplace=False)
    (3): Linear(in_features=256, out_features=128, bias=True)
    (4): ELU()
    (5): Dropout(p=0.07366997122359899, inplace=False)
    (6): Linear(in_features=128, out_features=128, bias=True)
    (7): ELU()
    (8): Dropout(p=0.07366997122359899, inplace=False)
    (9): Linear(in_features=128, out_features=64, bias=True)
    (10): ELU()
    (11): Dropout(p=0.07366997122359899, inplace=False)
    (12): Linear(in_features=64, out_features=11, bias=True)
  )
)
```

Validate metric	DataLoader 0
hp_metric	2.0352413654327393
val_acc	0.5070422291755676
val_loss	2.0352413654327393
valid_mapk	0.4923115670681

train_model result: {'valid_mapk': 0.4923115670681, 'val_loss': 2.0352413654327393, 'val_acc': 0.5070422291755676, 'valid_mapk': 0.4923115670681}

```

model: NetLightBase(
  (act_fn): ELU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=256, bias=True)
    (1): ELU()
    (2): Dropout(p=0.07366997122359899, inplace=False)
    (3): Linear(in_features=256, out_features=128, bias=True)
    (4): ELU()
    (5): Dropout(p=0.07366997122359899, inplace=False)
    (6): Linear(in_features=128, out_features=128, bias=True)
    (7): ELU()
    (8): Dropout(p=0.07366997122359899, inplace=False)
    (9): Linear(in_features=128, out_features=64, bias=True)
    (10): ELU()
    (11): Dropout(p=0.07366997122359899, inplace=False)
    (12): Linear(in_features=64, out_features=11, bias=True)
  )
)

```

Validate metric	DataLoader 0
hp_metric	1.9642229080200195
val_acc	0.577464759349823
val_loss	1.9642229080200195
valid_mapk	0.5972222685813904

train_model result: {'valid_mapk': 0.5972222685813904, 'val_loss': 1.9642229080200195, 'val_k: 4

```

model: NetLightBase(
  (act_fn): ELU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=256, bias=True)
    (1): ELU()
    (2): Dropout(p=0.07366997122359899, inplace=False)

```

```

(3): Linear(in_features=256, out_features=128, bias=True)
(4): ELU()
(5): Dropout(p=0.07366997122359899, inplace=False)
(6): Linear(in_features=128, out_features=128, bias=True)
(7): ELU()
(8): Dropout(p=0.07366997122359899, inplace=False)
(9): Linear(in_features=128, out_features=64, bias=True)
(10): ELU()
(11): Dropout(p=0.07366997122359899, inplace=False)
(12): Linear(in_features=64, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	1.8769389390945435
val_acc	0.6760563254356384
val_loss	1.8769389390945435
valid_mapk	0.6889880299568176

train_model result: {'valid_mapk': 0.6889880299568176, 'val_loss': 1.8769389390945435, 'val_acc': 0.6760563254356384, 'hp_metric': 1.8769389390945435}

```

model: NetLightBase(
  (act_fn): ELU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=256, bias=True)
    (1): ELU()
    (2): Dropout(p=0.07366997122359899, inplace=False)
    (3): Linear(in_features=256, out_features=128, bias=True)
    (4): ELU()
    (5): Dropout(p=0.07366997122359899, inplace=False)
    (6): Linear(in_features=128, out_features=128, bias=True)
    (7): ELU()
    (8): Dropout(p=0.07366997122359899, inplace=False)
    (9): Linear(in_features=128, out_features=64, bias=True)
    (10): ELU()
    (11): Dropout(p=0.07366997122359899, inplace=False)
  )
)

```

```

    (12): Linear(in_features=64, out_features=11, bias=True)
  )
)

```

Validate metric	DataLoader 0
hp_metric	1.832760214805603
val_acc	0.7183098793029785
val_loss	1.832760214805603
valid_mapk	0.7497520446777344

train_model result: {'valid_mapk': 0.7497520446777344, 'val_loss': 1.832760214805603, 'val_acc': 0.7183098793029785}

```

model: NetLightBase(
  (act_fn): ELU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=256, bias=True)
    (1): ELU()
    (2): Dropout(p=0.07366997122359899, inplace=False)
    (3): Linear(in_features=256, out_features=128, bias=True)
    (4): ELU()
    (5): Dropout(p=0.07366997122359899, inplace=False)
    (6): Linear(in_features=128, out_features=128, bias=True)
    (7): ELU()
    (8): Dropout(p=0.07366997122359899, inplace=False)
    (9): Linear(in_features=128, out_features=64, bias=True)
    (10): ELU()
    (11): Dropout(p=0.07366997122359899, inplace=False)
    (12): Linear(in_features=64, out_features=11, bias=True)
  )
)

```

Validate metric	DataLoader 0
hp_metric	1.8073997497558594
val_acc	0.7323943376541138

val_loss	1.8073997497558594
valid_mapk	0.7415674328804016

train_model result: {'valid_mapk': 0.7415674328804016, 'val_loss': 1.8073997497558594, 'val_k': 7}

```
model: NetLightBase(
  (act_fn): ELU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=256, bias=True)
    (1): ELU()
    (2): Dropout(p=0.07366997122359899, inplace=False)
    (3): Linear(in_features=256, out_features=128, bias=True)
    (4): ELU()
    (5): Dropout(p=0.07366997122359899, inplace=False)
    (6): Linear(in_features=128, out_features=128, bias=True)
    (7): ELU()
    (8): Dropout(p=0.07366997122359899, inplace=False)
    (9): Linear(in_features=128, out_features=64, bias=True)
    (10): ELU()
    (11): Dropout(p=0.07366997122359899, inplace=False)
    (12): Linear(in_features=64, out_features=11, bias=True)
  )
)
```

Validate metric	DataLoader 0
hp_metric	1.7426037788391113
val_acc	0.800000011920929
val_loss	1.7426037788391113
valid_mapk	0.8263888359069824

train_model result: {'valid_mapk': 0.8263888359069824, 'val_loss': 1.7426037788391113, 'val_k': 8}

```
model: NetLightBase(
  (act_fn): ELU()
  (train_mapk): MAPK()
```



```

(valid_mapk): MAPK()
(test_mapk): MAPK()
(layers): Sequential(
  (0): Linear(in_features=64, out_features=256, bias=True)
  (1): ELU()
  (2): Dropout(p=0.07366997122359899, inplace=False)
  (3): Linear(in_features=256, out_features=128, bias=True)
  (4): ELU()
  (5): Dropout(p=0.07366997122359899, inplace=False)
  (6): Linear(in_features=128, out_features=128, bias=True)
  (7): ELU()
  (8): Dropout(p=0.07366997122359899, inplace=False)
  (9): Linear(in_features=128, out_features=64, bias=True)
  (10): ELU()
  (11): Dropout(p=0.07366997122359899, inplace=False)
  (12): Linear(in_features=64, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	1.7411670684814453
val_acc	0.800000011920929
val_loss	1.7411670684814453
valid_mapk	0.8090277314186096

train_model result: {'valid_mapk': 0.8090277314186096, 'val_loss': 1.7411670684814453, 'val_acc': 0.8, 'k': 9}

```

model: NetLightBase(
  (act_fn): ELU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=256, bias=True)
    (1): ELU()
    (2): Dropout(p=0.07366997122359899, inplace=False)
    (3): Linear(in_features=256, out_features=128, bias=True)
    (4): ELU()
    (5): Dropout(p=0.07366997122359899, inplace=False)

```

```

(6): Linear(in_features=128, out_features=128, bias=True)
(7): ELU()
(8): Dropout(p=0.07366997122359899, inplace=False)
(9): Linear(in_features=128, out_features=64, bias=True)
(10): ELU()
(11): Dropout(p=0.07366997122359899, inplace=False)
(12): Linear(in_features=64, out_features=11, bias=True)
)
)

```

Validate metric	DataLoader 0
hp_metric	1.7557319402694702
val_acc	0.7857142686843872
val_loss	1.7557319402694702
valid_mapk	0.8020833134651184

```

train_model result: {'valid_mapk': 0.8020833134651184, 'val_loss': 1.7557319402694702, 'val_
cv_model mapk result: 0.6702628910541535

```

```

0.6702628910541535

```

i Note: Evaluation for the Final Comaprison

- 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)

```

```

epochs: 0.07735633638265811
batch_size: 0.14004188316518415
optimizer: 8.157773046164346
patience: 100.0
initialization: 51.76537839159858

```

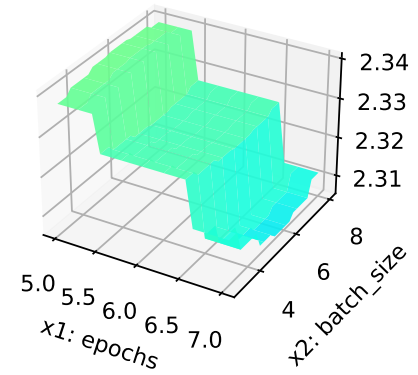
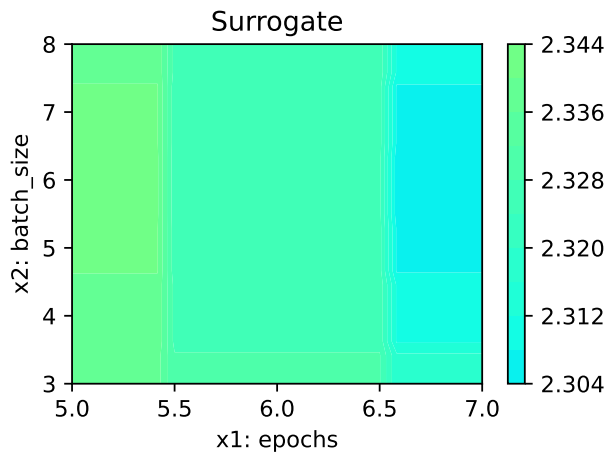
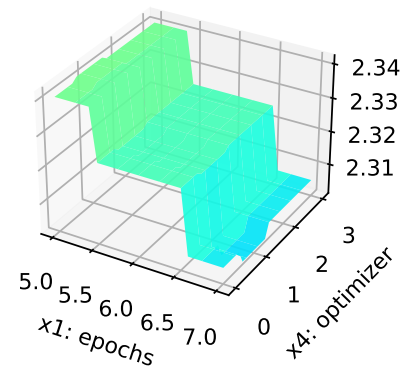
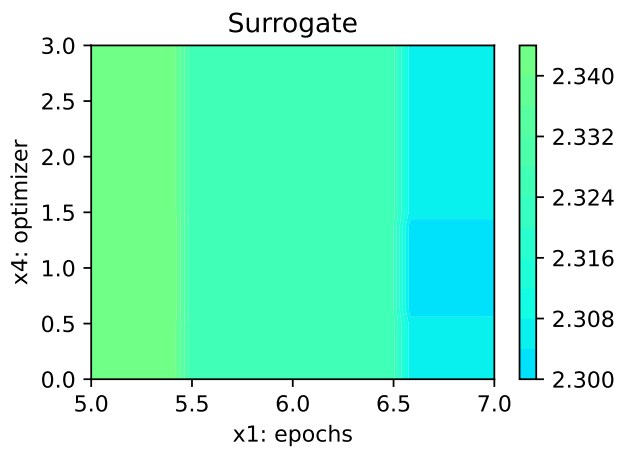
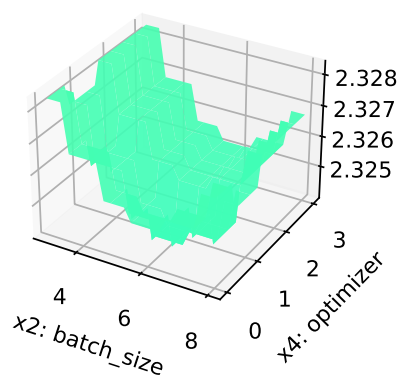
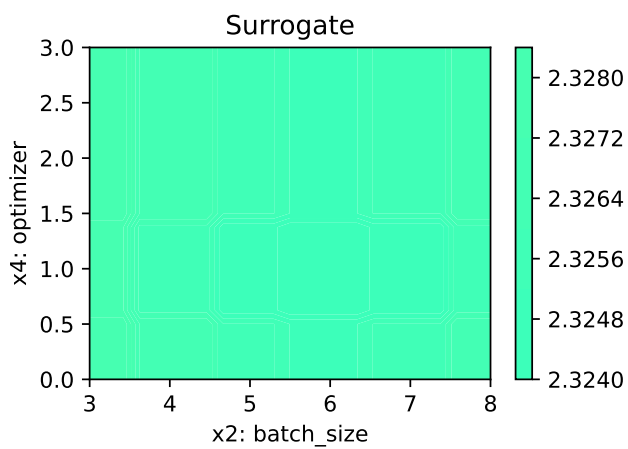
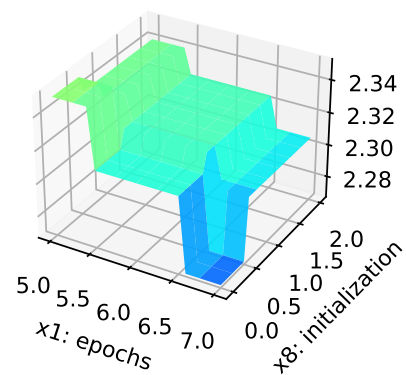
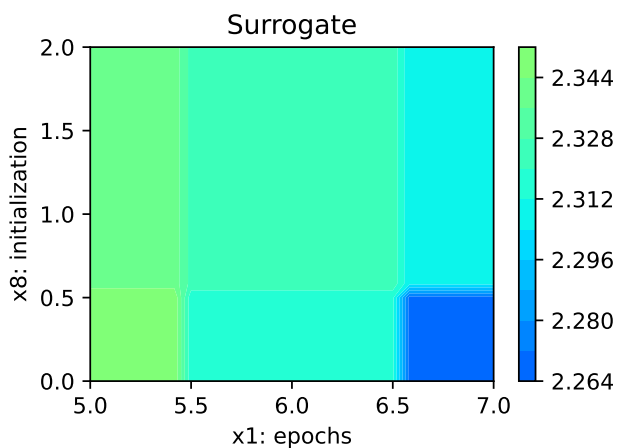
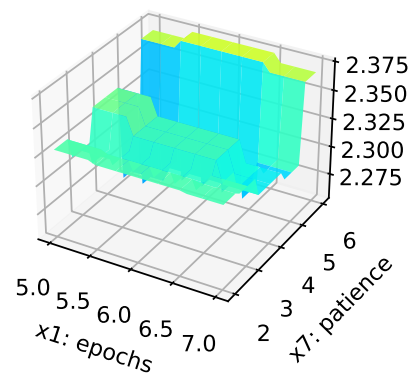
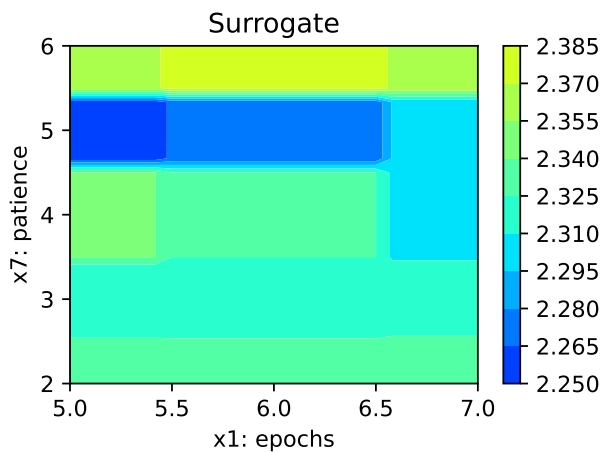
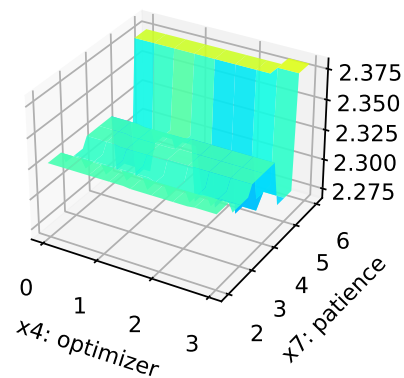
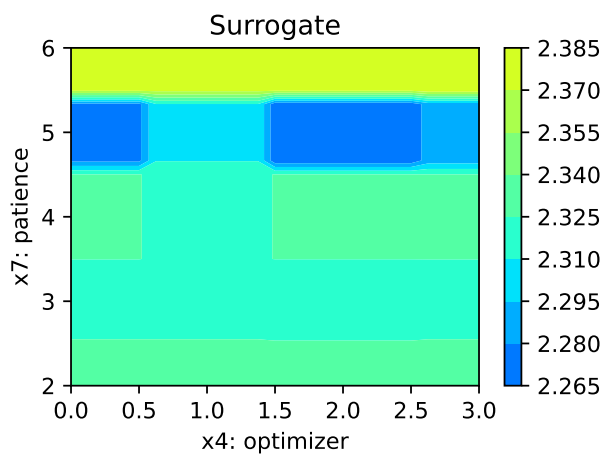
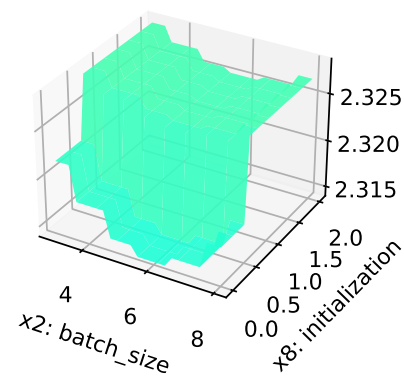
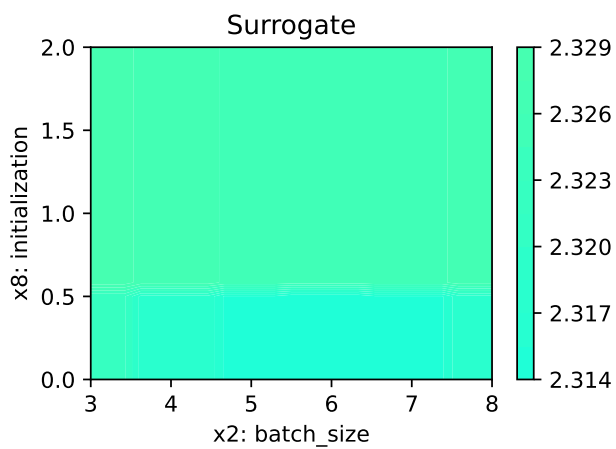
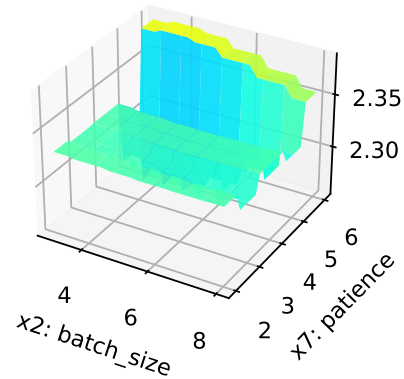
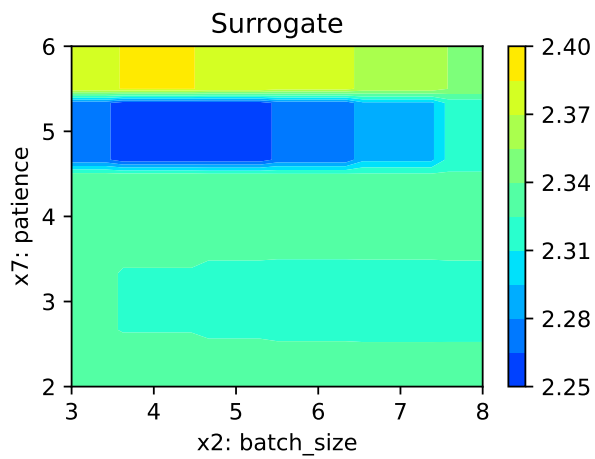
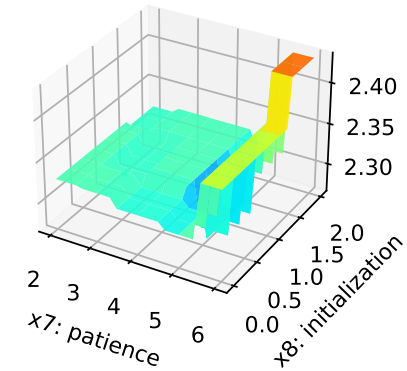
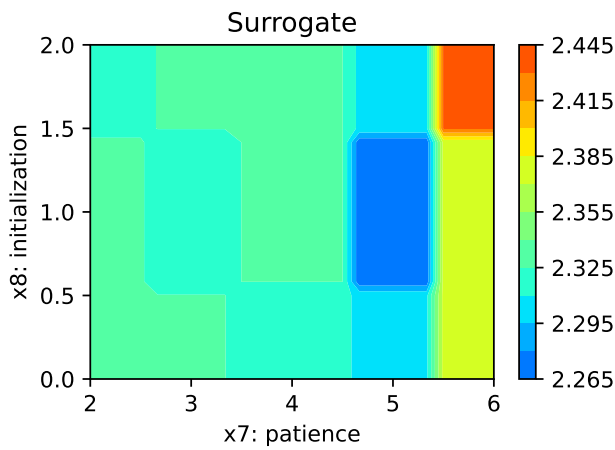
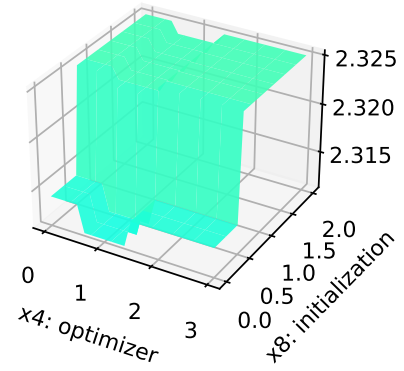
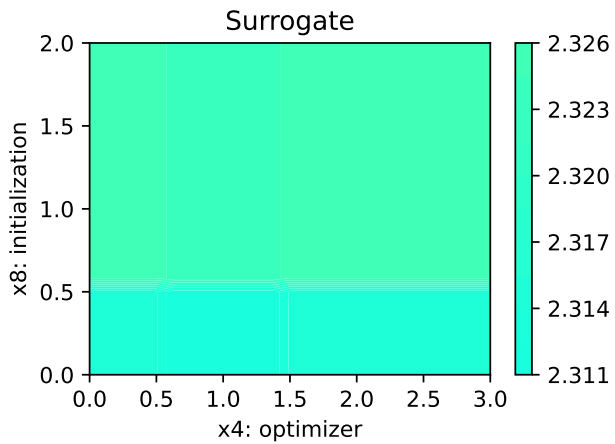


Figure 21.3: Contour plots.









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(
  (act_fn): ELU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
```

```

(0): Linear(in_features=64, out_features=256, bias=True)
(1): ELU()
(2): Dropout(p=0.07366997122359899, inplace=False)
(3): Linear(in_features=256, out_features=128, bias=True)
(4): ELU()
(5): Dropout(p=0.07366997122359899, inplace=False)
(6): Linear(in_features=128, out_features=128, bias=True)
(7): ELU()
(8): Dropout(p=0.07366997122359899, inplace=False)
(9): Linear(in_features=128, out_features=64, bias=True)
(10): ELU()
(11): Dropout(p=0.07366997122359899, inplace=False)
(12): Linear(in_features=64, out_features=11, bias=True)
)
)

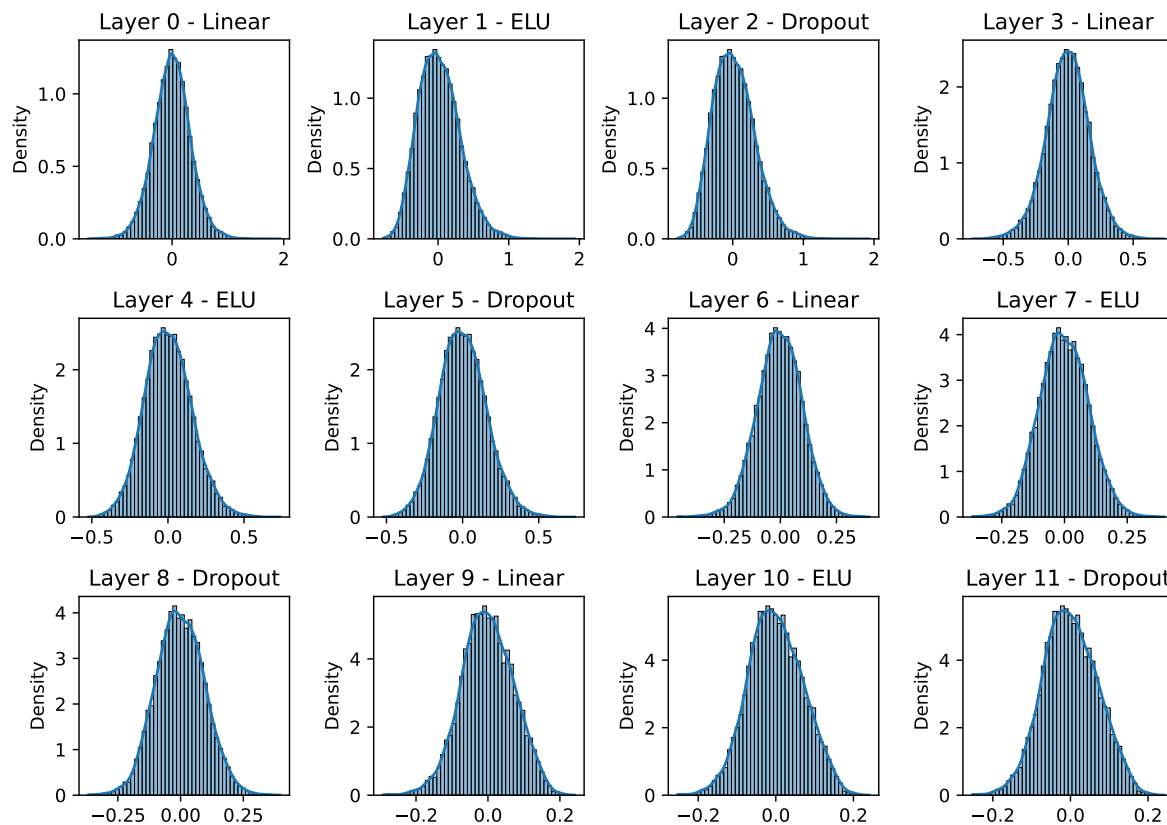
```

```

from spotPython.utils.eda import visualize_activations
visualize_activations(model, device="cpu", color=f"C{0}")

```


Activation distribution for activation function ELU()



22 Documentation of the Sequential Parameter Optimization

This document describes the `Spot` features.

22.1 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
```

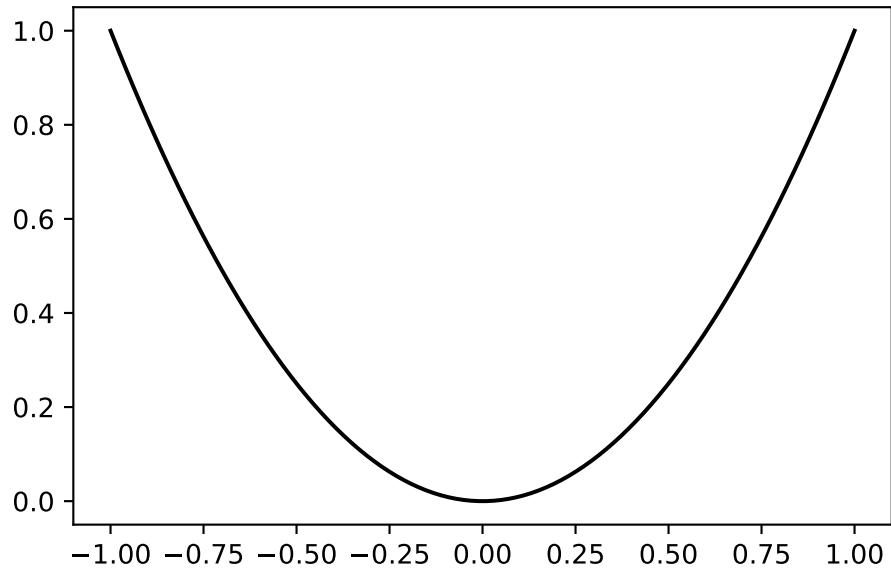
22.1.1 The Objective Function

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`.

22.1.2 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`

22.2 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

22.3 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

22.4 The surrogate_control Dictionary

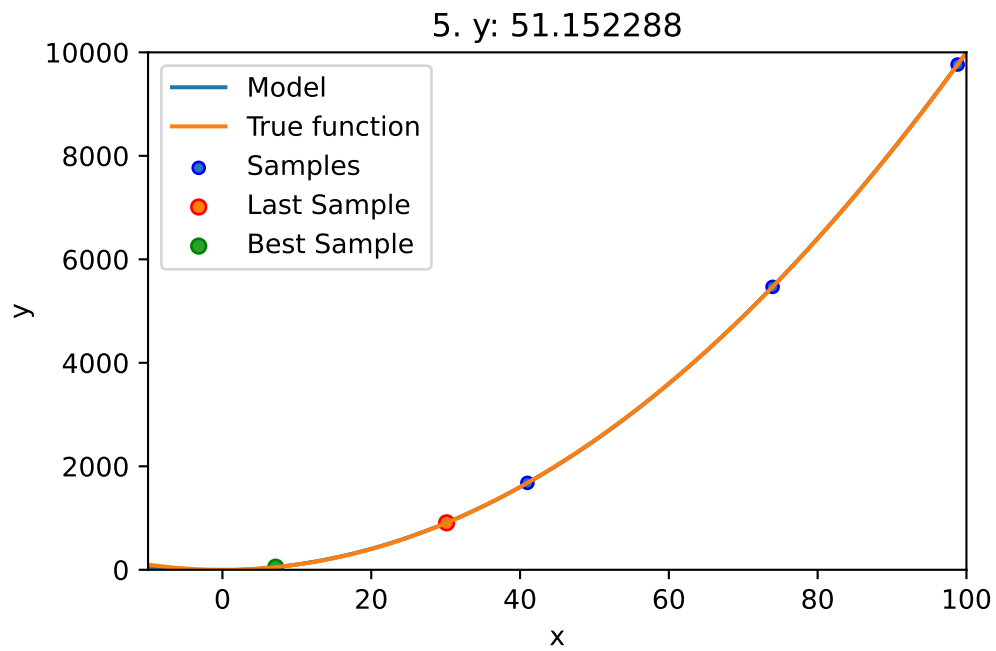
external parameter	type	description	default	mandatory
noise				
model_optimizer	object	optimizer	differential_evolution	
model_fun_evals				
min_theta			-3.	
max_theta			3.	
n_theta			1	
n_p			1	
optim_p			False	
cod_type			"norm"	
var_type				
use_cod_y	bool		False	

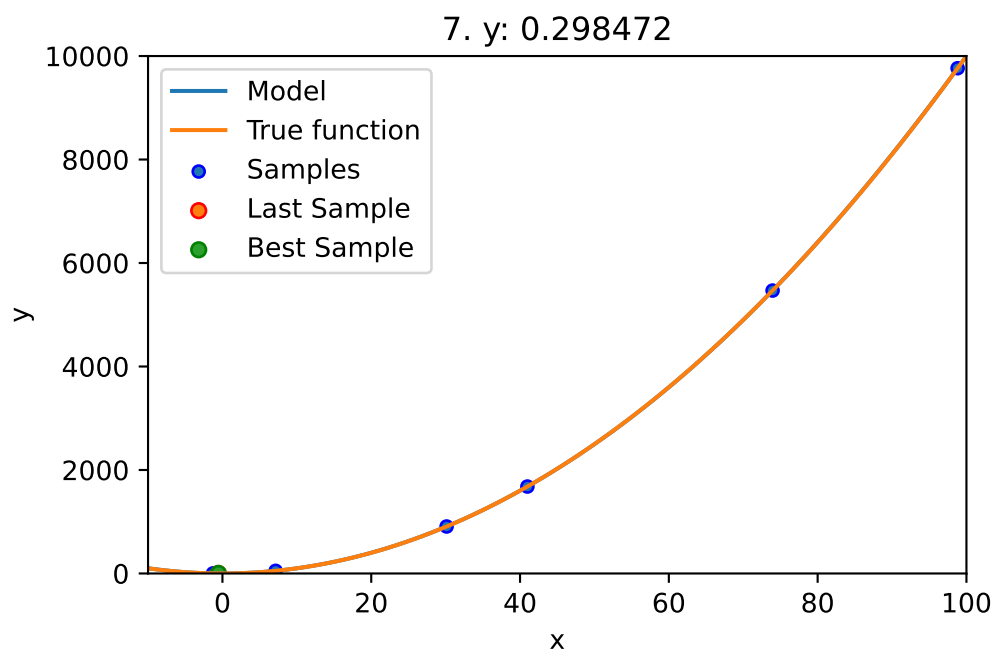
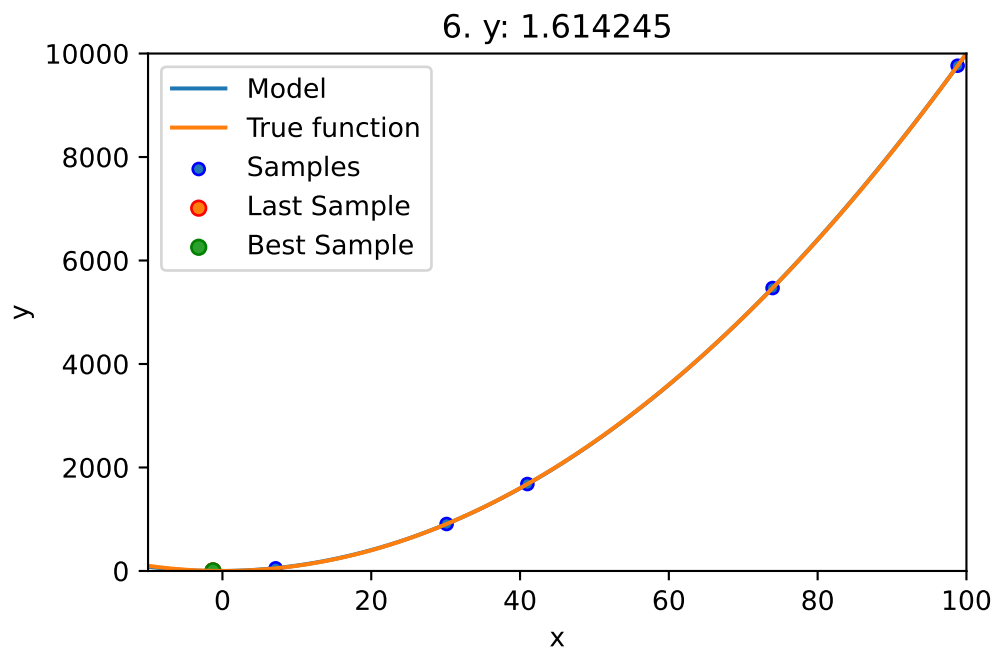
22.5 The optimizer_control Dictionary

external parameter	type	description	default	mandatory
max_iter	int	max number of iterations. Note: these are the cheap evaluations on the surrogate.	1000	no

22.6 Run

```
spot_1.run()
```





<spotPython.spot.spot.Spot at 0x161743df0>

22.7 Print the Results

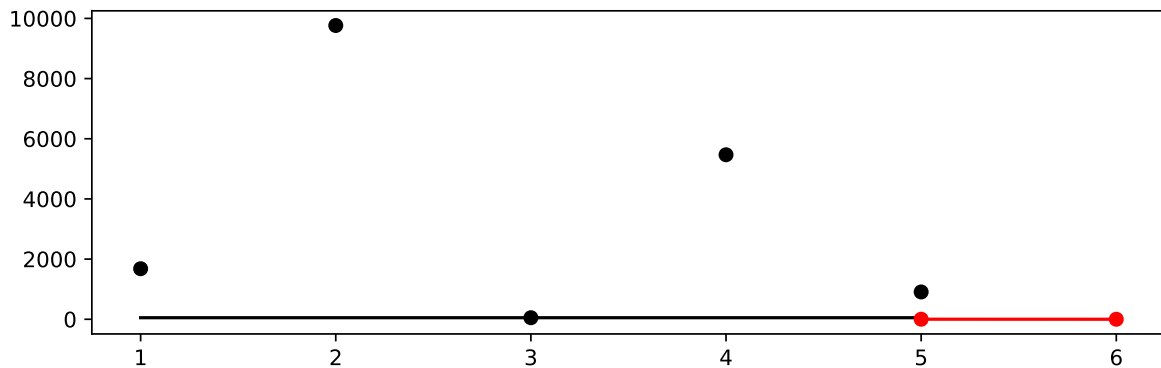
```
spot_1.print_results()
```

```
min y: 0.29847171516431314  
x0: -0.5463256493743572
```

```
[['x0', -0.5463256493743572]]
```

22.8 Show the Progress

```
spot_1.plot_progress()
```

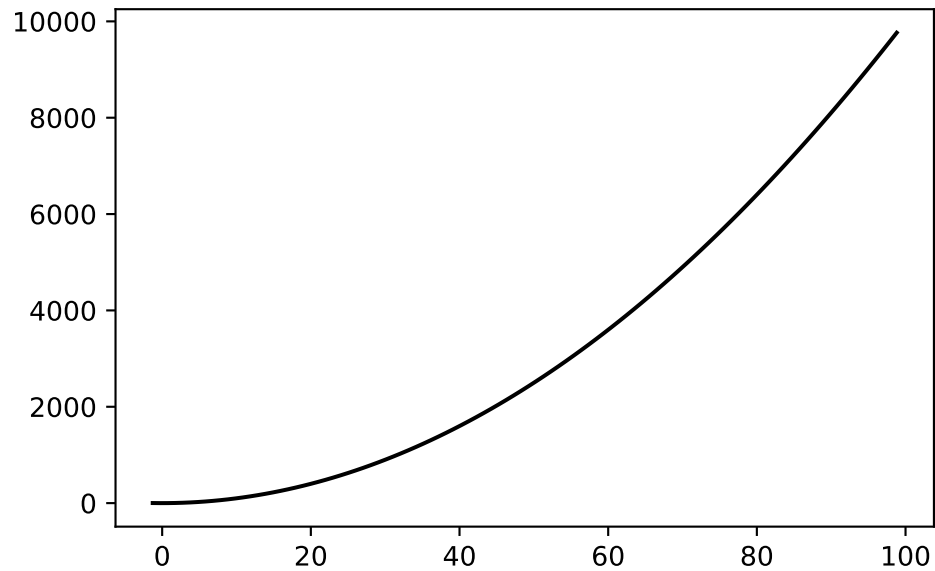


22.9 Visualize the Surrogate

- 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>



22.10 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]

```

22.11 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 ]]))

```

22.12 Surrogates

22.12.1 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 .

22.13 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          nan  0.78240941 -0.58120945
 -0.3923345   0.67234256  0.31802454          nan -0.75129705  0.97550354
  0.41757584          nan  0.82585329  0.23700598 -0.49274073 -0.82319082
 -0.17991251  0.1481835 ]

```

```

[nan]
[-1.]

```

```

[-0.47259301]
[0.95541987]

```

```

[0.17335968]
[nan]

```

```

[-0.58552368]
[-0.20126111]

```

```

[-0.60100809]
[nan]

```

```

[-0.97897336]
[-0.2748985]

```

[nan]
[0.8359486]

[0.99035591]
[0.01641232]

[0.5629346]

22.14 PyTorch: Detailed Description of the Data Splitting

22.14.1 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

```

22.14.1.1 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.

22.14.1.2 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`.

22.14.1.3 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.

22.14.1.4 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.