# Hyperparameter Tuning Cookbook

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

Thomas Bartz-Beielstein

Jul 6, 2023

# Table of contents


**Preface** **12**
Citation . . . 12

**1 Introduction: Hyperparameter Tuning** **14**
1.1 The Hyperparameter Tuning Software SPOT . . . 15
1.2 Spot as an Optimizer . . . 16
1.3 Example: `Spot` and the Sphere Function . . . 17
1.3.1 The Objective Function: Sphere . . . 17
1.4 `Spot` Parameters: `fun_evals`, `init_size` and `show_models` . . . 19
1.5 Print the Results . . . 21
1.6 Show the Progress . . . 21

**2 Multi-dimensional Functions** **23**
2.1 Example: `Spot` and the 3-dim Sphere Function . . . 23
2.1.1 The Objective Function: 3-dim Sphere . . . 23
2.1.2 Results . . . 24
2.1.3 A Contour Plot . . . 25
2.2 Conclusion . . . 27
2.3 Exercises . . . 27
2.3.1 The Three Dimensional `fun_cubed` . . . 27
2.3.2 The Ten Dimensional `fun_wing_wt` . . . 28
2.3.3 The Three Dimensional `fun_runge` . . . 28
2.3.4 The Three Dimensional `fun_linear` . . . 28

**3 Isotropic and Anisotropic Kriging** **29**
3.1 Example: Isotropic `Spot` Surrogate and the 2-dim Sphere Function . . . 29
3.1.1 The Objective Function: 2-dim Sphere . . . 29
3.1.2 Results . . . 30
3.2 Example With Anisotropic Kriging . . . 30
3.2.1 Taking a Look at the `theta` Values . . . 31
3.3 Exercises . . . 32
3.3.1 `fun_branin` . . . 32
3.3.2 `fun_sin_cos` . . . 33
3.3.3 `fun_runge` . . . 33
3.3.4 `fun_wingwt` . . . 33

**4 Using `sklearn` Surrogates in `spotPython` 34**
4.1 Example: Branin Function with `spotPython`'s Internal Kriging Surrogate . . . 34
4.1.1 The Objective Function Branin . . . 34
4.1.2 Running the surrogate model based optimizer `Spot`: . . . 35
4.1.3 Print the Results . . . 35
4.1.4 Show the Progress and the Surrogate . . . 35
4.2 Example: Using Surrogates From scikit-learn . . . 36
4.2.1 GaussianProcessRegressor as a Surrogate . . . 37
4.3 Example: One-dimensional Sphere Function With `spotPython`'s Kriging . . . 39
4.3.1 Results . . . 44
4.4 Example: `Sklearn` Model GaussianProcess . . . 45
4.5 Exercises . . . 51
4.5.1 `DecisionTreeRegressor` . . . 51
4.5.2 `RandomForestRegressor` . . . 51
4.5.3 `linear_model.LinearRegression` . . . 51
4.5.4 `linear_model.Ridge` . . . 52
4.6 Exercise 2 . . . 52

**5 Sequential Parameter Optimization: Using `scipy` Optimizers 53**
5.1 The Objective Function Branin . . . 53
5.2 The Optimizer . . . 54
5.3 Print the Results . . . 55
5.4 Show the Progress . . . 55
5.5 Exercises . . . 56
5.5.1 `dual_annealing` . . . 56
5.5.2 `direct` . . . 56
5.5.3 `shgo` . . . 57
5.5.4 `basinhopping` . . . 57
5.5.5 Performance Comparison . . . 57

**6 Sequential Parameter Optimization: Gaussian Process Models 58**
6.1 Gaussian Processes Regression: Basic Introductory `scikit-learn` Example . . 58
6.1.1 Train and Test Data . . . 59
6.1.2 Building the Surrogate With `Sklearn` . . . 59
6.1.3 Plotting the `Sklearn`Model . . . 59
6.1.4 The `spotPython` Version . . . 60
6.1.5 Visualizing the Differences Between the `spotPython` and the `sklearn` Model Fits . . . 61
6.2 Exercises . . . 62
6.2.1 `Schonlau Example Function` . . . 62
6.2.2 `Forrester Example Function` . . . 62
6.2.3 `fun_runge Function (1-dim)` . . . 63
6.2.4 `fun_cubed (1-dim)` . . . 64

6.2.5 The Effect of Noise . . . 64

**7 Expected Improvement** **66**
7.1 Example: `Spot` and the 1-dim Sphere Function . . . 66
7.1.1 The Objective Function: 1-dim Sphere . . . 66
7.1.2 Results . . . 67
7.2 Same, but with EI as infill_criterion . . . 67
7.3 Non-isotropic Kriging . . . 68
7.4 Using `sklearn` Surrogates . . . 70
7.4.1 The spot Loop . . . 70
7.4.2 spot: The Initial Model . . . 72
7.4.3 Init: Build Initial Design . . . 72
7.4.4 Evaluate . . . 75
7.4.5 Build Surrogate . . . 75
7.4.6 A Simple Predictor . . . 75
7.5 Gaussian Processes regression: basic introductory example . . . 75
7.6 The Surrogate: Using scikit-learn models . . . 78
7.7 Additional Examples . . . 80
7.7.1 Optimize on Surrogate . . . 84
7.7.2 Evaluate on Real Objective . . . 84
7.7.3 Impute / Infill new Points . . . 84
7.8 Tests . . . 84
7.9 EI: The Famous Schonlau Example . . . 85
7.10 EI: The Forrester Example . . . 87
7.11 Noise . . . 90
7.12 Cubic Function . . . 93
7.13 Factors . . . 99

**8 Hyperparameter Tuning and Noise** **101**
8.1 Example: `Spot` and the Noisy Sphere Function . . . 101
8.1.1 The Objective Function: Noisy Sphere . . . 101
8.2 Print the Results . . . 105
8.3 Noise and Surrogates: The Nugget Effect . . . 105
8.3.1 The Noisy Sphere . . . 105
8.4 Exercises . . . 108
8.4.1 Noisy `fun_cubed` . . . 108
8.4.2 `fun_runge` . . . 109
8.4.3 `fun_forrester` . . . 109
8.4.4 `fun_xsin` . . . 109

**9 Handling Noise: Optimal Computational Budget Allocation in `Spot`** **110**
9.1 Example: `Spot`, OCBA, and the Noisy Sphere Function . . . 110
9.1.1 The Objective Function: Noisy Sphere . . . 110

9.2 Print the Results . . . 120
9.3 Noise and Surrogates: The Nugget Effect . . . 121
9.3.1 The Noisy Sphere . . . 121
9.4 Exercises . . . 124
9.4.1 Noisy `fun_cubed` . . . 124
9.4.2 `fun_runge` . . . 124
9.4.3 `fun_forrester` . . . 124
9.4.4 `fun_xsin` . . . 125

**10 HPT: sklearn SVC on Moons Data** **126**
10.1 Step 1: Setup . . . 126
10.2 Step 2: Initialization of the Empty `fun_control` Dictionary . . . 127
10.3 Step 3: SKlearn Load Data (Classification) . . . 127
10.4 Step 4: Specification of the Preprocessing Model . . . 129
10.5 Step 5: Select Model (`algorithm`) and `core_model_hyper_dict` . . . 130
10.6 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model` . . . 132
10.6.1 Modify hyperparameter of type numeric and integer (boolean) . . . 133
10.6.2 Modify hyperparameter of type factor . . . 133
10.6.3 Optimizers . . . 133
10.7 Step 7: Selection of the Objective (Loss) Function . . . 134
10.7.1 Predict Classes or Class Probabilities . . . 134
10.8 Step 8: Calling the SPOT Function . . . 134
10.8.1 Preparing the SPOT Call . . . 134
10.8.2 The Objective Function . . . 135
10.8.3 Run the `Spot` Optimizer . . . 135
10.8.4 Starting the Hyperparameter Tuning . . . 136
10.9 Step 9: Results . . . 137
10.9.1 Show variable importance . . . 139
10.9.2 Get Default Hyperparameters . . . 139
10.9.3 Get SPOT Results . . . 140
10.9.4 Plot: Compare Predictions . . . 141
10.9.5 Detailed Hyperparameter Plots . . . 143
10.9.6 Parallel Coordinates Plot . . . 144
10.9.7 Plot all Combinations of Hyperparameters . . . 144

**11 HPT: River** **145**
11.1 Step 1: Setup . . . 145
11.1.1 `river` Hyperparameter Tuning: HATR with Friedman Drift Data . . . 145
11.2 Step 2: Initialization of the `fun_control` Dictionary . . . 146
11.3 Step 3: Load the Friedman Drift Data . . . 146
11.4 Step 4: Specification of the Preprocessing Model . . . 147
11.5 Step 5: Select `algorithm` and `core_model_hyper_dict` . . . 148

11.6 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model` . . . 151
11.6.1 Modify hyperparameter of type factor . . . 151
11.6.2 Modify hyperparameter of type numeric and integer (boolean) . . . 151
11.7 Step 7: Selection of the Objective (Loss) Function . . . 151
11.8 Step 8: Calling the SPOT Function . . . 152
11.8.1 Prepare the SPOT Parameters . . . 152
11.8.2 Run the `Spot` Optimizer . . . 153
11.9 Step 9: Results . . . 154
11.9.1 Show variable importance . . . 156
11.9.2 Build and Evaluate HTR Model with Tuned Hyperparameters . . . 156
11.9.3 The Large Data Set (k=0.2) . . . 156
11.9.4 Get Default Hyperparameters . . . 157
11.9.5 Get SPOT Results . . . 160
11.9.6 Visualize Regression Trees . . . 164
11.9.7 Spot Model . . . 165
11.9.8 Detailed Hyperparameter Plots . . . 166
11.9.9 Parallel Coordinates Plots . . . 170
11.9.10 Plot all Combinations of Hyperparameters . . . 170

**12 HPT: PyTorch With `spotPython` and Ray Tune on CIFAR10 171**
12.1 Step 1: Setup . . . 171
12.2 Step 2: Initialization of the `fun_control` Dictionary . . . 173
12.3 Step 3: PyTorch Data Loading . . . 173
12.4 Step 4: Specification of the Preprocessing Model . . . 174
12.5 Step 5: Select Model (`algorithm`) and `core_model_hyper_dict` . . . 174
12.5.1 The `Net_Core` class . . . 176
12.5.2 Comparison of the Approach Described in the PyTorch Tutorial With spotPython . . . 177
12.5.3 The Search Space: Hyperparameters . . . 178
12.5.4 Configuring the Search Space With Ray Tune . . . 178
12.5.5 Configuring the Search Space With spotPython . . . 178
12.6 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model` . . . 180
12.6.1 Optimizers . . . 182
12.7 Step 7: Selection of the Objective (Loss) Function . . . 184
12.7.1 Evaluation: Data Splitting . . . 184
12.7.2 Hold-out Data Split . . . 184
12.7.3 Cross-Validation . . . 185
12.7.4 Overview of the Evaluation Settings . . . 185
12.7.5 Evaluation: Loss Functions and Metrics . . . 186
12.8 Step 8: Calling the SPOT Function . . . 187
12.8.1 Preparing the SPOT Call . . . 187

12.8.2 The Objective Function `fun_torch` . . . 188
12.8.3 Using Default Hyperparameters or Results from Previous Runs . . . 188
12.8.4 Starting the Hyperparameter Tuning . . . 189
12.9 Step 9: Tensorboard . . . 193
12.9.1 Tensorboard: Start Tensorboard . . . 193
12.9.2 Saving the State of the Notebook . . . 195
12.10 Step 10: Results . . . 195
12.10.1 Get the Tuned Architecture (SPOT Results) . . . 197
12.10.2 Get Default Hyperparameters . . . 198
12.10.3 Evaluation of the Default Architecture . . . 198
12.10.4 Evaluation of the Tuned Architecture . . . 200
12.10.5 Detailed Hyperparameter Plots . . . 202
12.11 Summary and Outlook . . . 206
12.12 Appendix . . . 207
12.12.1 Sample Output From Ray Tune's Run . . . 207

**13 HPT: sklearn RandomForestClassifier VBDP Data 208**
13.1 Step 1: Setup . . . 208
13.2 Step 2: Initialization of the Empty `fun_control` Dictionary . . . 209
13.3 Step 3: PyTorch Data Loading . . . 210
13.3.1 Load Data: Classification VBDP . . . 210
13.3.2 Holdout Train and Test Data . . . 210
13.4 Step 4: Specification of the Preprocessing Model . . . 211
13.5 Step 5: Select Model (`algorithm`) and `core_model_hyper_dict` . . . 212
13.6 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model` . . . 214
13.6.1 Modify hyperparameter of type numeric and integer (boolean) . . . 214
13.6.2 Modify hyperparameter of type factor . . . 214
13.6.3 Optimizers . . . 215
13.6.4 Selection of the Objective: Metric and Loss Functions . . . 215
13.7 Step 7: Selection of the Objective (Loss) Function . . . 215
13.7.1 Metric Function . . . 215
13.7.2 Evaluation on Hold-out Data . . . 216
13.7.3 OOB Score . . . 216
13.8 Step 8: Calling the SPOT Function . . . 217
13.8.1 Preparing the SPOT Call . . . 217
13.8.2 The Objective Function . . . 218
13.8.3 Run the `Spot` Optimizer . . . 218
13.9 Step 9: Tensorboard . . . 221
13.10 Step 10: Results . . . 221
13.10.1 Show variable importance . . . 222
13.10.2 Get Default Hyperparameters . . . 223
13.10.3 Get SPOT Results . . . 224

13.10.4 Evaluate SPOT Results . . . 225
13.10.5 Handling Non-deterministic Results . . . 226
13.10.6 Evalution of the Default Hyperparameters . . . 226
13.10.7 Plot: Compare Predictions . . . 227
13.10.8 Cross-validated Evaluations . . . 228
13.10.9 Detailed Hyperparameter Plots . . . 229
13.10.10 Parallel Coordinates Plot . . . 233
13.10.11 Plot all Combinations of Hyperparameters . . . 233

**14 HPT: sklearn XGB Classifier VBDP Data 234**
14.1 Step 1: Setup . . . 234
14.2 Step 2: Initialization of the Empty `fun_control` Dictionary . . . 235
14.3 Step 3: PyTorch Data Loading . . . 236
14.3.1 1. Load Data: Classification VBDP . . . 236
14.3.2 Holdout Train and Test Data . . . 236
14.4 Step 4: Specification of the Preprocessing Model . . . 237
14.5 Step 5: Select Model (`algorithm`) and `core_model_hyper_dict` . . . 238
14.6 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model` . . . 240
14.6.1 Modify hyperparameter of type numeric and integer (boolean) . . . 240
14.6.2 Modify hyperparameter of type factor . . . 240
14.6.3 Optimizers . . . 241
14.7 Step 7: Selection of the Objective (Loss) Function . . . 241
14.7.1 Evaluation . . . 241
14.7.2 Selection of the Objective: Metric and Loss Functions . . . 241
14.7.3 Loss Function . . . 241
14.7.4 Metric Function . . . 241
14.7.5 Evaluation on Hold-out Data . . . 242
14.8 Step 8: Calling the SPOT Function . . . 243
14.8.1 Preparing the SPOT Call . . . 243
14.8.2 The Objective Function . . . 244
14.8.3 Run the `Spot` Optimizer . . . 244
14.9 Step 9: Tensorboard . . . 246
14.10 Step 10: Results . . . 246
14.10.1 Show variable importance . . . 247
14.10.2 Get Default Hyperparameters . . . 247
14.10.3 Get SPOT Results . . . 248
14.10.4 Evaluate SPOT Results . . . 249
14.10.5 Handling Non-deterministic Results . . . 250
14.10.6 Evalution of the Default Hyperparameters . . . 250
14.10.7 Plot: Compare Predictions . . . 251
14.10.8 Cross-validated Evaluations . . . 253
14.10.9 Detailed Hyperparameter Plots . . . 254

14.10.10 Parallel Coordinates Plot . . . 257
14.10.11 Plot all Combinations of Hyperparameters . . . 257

**15 HPT: sklearn SVC VBDP Data** **258**
15.1 Step 1: Setup . . . 258
15.2 Step 2: Initialization of the Empty `fun_control` Dictionary . . . 259
15.3 Step 3: PyTorch Data Loading . . . 260
15.3.1 1. Load Data: Classification VBDP . . . 260
15.3.2 Holdout Train and Test Data . . . 260
15.4 Step 4: Specification of the Preprocessing Model . . . 261
15.5 Step 5: Select Model (`algorithm`) and `core_model_hyper_dict` . . . 262
15.6 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model` . . . 264
15.6.1 Modify hyperparameter of type numeric and integer (boolean) . . . 264
15.6.2 Modify hyperparameter of type factor . . . 264
15.6.3 Optimizers . . . 264
15.6.4 Selection of the Objective: Metric and Loss Functions . . . 265
15.7 Step 7: Selection of the Objective (Loss) Function . . . 265
15.7.1 Metric Function . . . 265
15.7.2 Evaluation on Hold-out Data . . . 266
15.8 Step 8: Calling the SPOT Function . . . 267
15.8.1 Preparing the SPOT Call . . . 267
15.8.2 The Objective Function . . . 268
15.8.3 Run the `Spot` Optimizer . . . 268
15.9 Step 9: Tensorboard . . . 272
15.10 Step 10: Results . . . 273
15.10.1 Show variable importance . . . 274
15.10.2 Get Default Hyperparameters . . . 274
15.10.3 Get SPOT Results . . . 275
15.10.4 Evaluate SPOT Results . . . 276
15.10.5 Handling Non-deterministic Results . . . 277
15.10.6 Evalution of the Default Hyperparameters . . . 277
15.10.7 Plot: Compare Predictions . . . 278
15.10.8 Cross-validated Evaluations . . . 280
15.10.9 Detailed Hyperparameter Plots . . . 281
15.10.10 Parallel Coordinates Plot . . . 282
15.10.11 Plot all Combinations of Hyperparameters . . . 282

**16 HPT: sklearn KNN Classifier VBDP Data** **284**
16.1 Step 1: Setup . . . 284
16.2 Step 2: Initialization of the Empty `fun_control` Dictionary . . . 285
16.2.1 Load Data: Classification VBDP . . . 285
16.2.2 Holdout Train and Test Data . . . 286

16.3 Step 4: Specification of the Preprocessing Model . . . 287
16.4 Step 5: Select Model (`algorithm`) and `core_model_hyper_dict` . . . 288
16.5 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model` . . . 289
16.5.1 Modify hyperparameter of type numeric and integer (boolean) . . . 289
16.5.2 Modify hyperparameter of type factor . . . 290
16.5.3 Optimizers . . . 290
16.5.4 Selection of the Objective: Metric and Loss Functions . . . 290
16.6 Step 7: Selection of the Objective (Loss) Function . . . 290
16.6.1 Metric Function . . . 291
16.6.2 Evaluation on Hold-out Data . . . 292
16.7 Step 8: Calling the SPOT Function . . . 292
16.7.1 Preparing the SPOT Call . . . 292
16.7.2 The Objective Function . . . 293
16.7.3 Run the `Spot` Optimizer . . . 293
16.8 Step 9: Tensorboard . . . 297
16.9 Step 10: Results . . . 297
16.9.1 Show variable importance . . . 298
16.9.2 Get Default Hyperparameters . . . 299
16.9.3 Get SPOT Results . . . 300
16.9.4 Evaluate SPOT Results . . . 300
16.9.5 Handling Non-deterministic Results . . . 301
16.9.6 Evalution of the Default Hyperparameters . . . 302
16.9.7 Plot: Compare Predictions . . . 302
16.9.8 Cross-validated Evaluations . . . 304
16.9.9 Detailed Hyperparameter Plots . . . 305
16.9.10 Parallel Coordinates Plot . . . 308
16.9.11 Plot all Combinations of Hyperparameters . . . 308

**17 HPT PyTorch Lightning: VBDP 309**
17.1 Step 1: Setup . . . 309
17.2 Step 2: Initialization of the `fun_control` Dictionary . . . 310
17.3 Step 3: PyTorch Data Loading . . . 311
17.3.1 Lightning Dataset and DataModule . . . 311
17.4 Step 4: Preprocessing . . . 311
17.5 Step 5: Select the NN Model (`algorithm`) and `core_model_hyper_dict` . . . 311
17.6 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model` . . . 312
17.7 Step 7: Data Splitting, the Objective (Loss) Function and the Metric . . . 314
17.7.1 Evaluation . . . 314
17.7.2 Loss Functions and Metrics . . . 314
17.7.3 Metric . . . 314

17.8 Step 8: Calling the SPOT Function . . . 315
17.8.1 Preparing the SPOT Call . . . 315
17.8.2 The Objective Function `fun` . . . 316
17.8.3 Starting the Hyperparameter Tuning . . . 316
17.9 Step 9: Tensorboard . . . 338
17.10 Step 10: Results . . . 338
17.10.1 Get the Tuned Architecture . . . 340
17.10.2 Cross Validation With Lightning . . . 341
17.10.3 Detailed Hyperparameter Plots . . . 350
17.10.4 Parallel Coordinates Plot . . . 351
17.10.5 Plot all Combinations of Hyperparameters . . . 351
17.10.6 Visualizing the Activation Distribution . . . 352
17.11 Submission . . . 353
17.12 Appendix . . . 355
17.12.1 Differences to the spotPython Approaches for `torch`, `sklearn` and `river` 355
17.13 Specification of the Preprocessing Model . . . 355
17.13.1 Taking a Look at the Data . . . 356

**18 Documentation of the Sequential Parameter Optimization 358**
18.1 Example: spot . . . 358
18.1.1 The Objective Function . . . 358
18.1.2 External Parameters . . . 360
18.2 The `fun_control` Dictionary . . . 363
18.3 The `design_control` Dictionary . . . 363
18.4 The `surrogate_control` Dictionary . . . 364
18.5 The `optimizer_control` Dictionary . . . 364
18.6 Run . . . 365
18.7 Print the Results . . . 367
18.8 Show the Progress . . . 367
18.9 Visualize the Surrogate . . . 367
18.10 Init: Build Initial Design . . . 368
18.11 Replicability . . . 369
18.12 Surrogates . . . 370
18.12.1 A Simple Predictor . . . 370
18.13 Demo/Test: Objective Function Fails . . . 370
18.14 PyTorch: Detailed Description of the Data Splitting . . . 372
18.14.1 Description of the `"train_hold_out"` Setting . . . 372

**References 383**

# 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 Intell
         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 github[1], allowing replicability of the results. This tutorial descries 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 12 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 12.1 describes the setup of the tuners. Section 12.3 describes the data loading. Section 12.5 describes the model to be tuned. The search space is introduced in Section 12.5.3. Optimizers are presented in Section 12.6.1. How to split the data in train, validation, and test sets is described in Section 12.7.1. The selection of the loss function and metrics is described in Section 12.7.5. Section 12.8.1 describes the preparation of the `spotPython` call. The objective function is described in Section 12.8.2. How to use results from previous runs and default hyperparameter configurations is described in Section 12.8.3. Starting the tuner is shown in Section 12.8.4. TensorBoard can be used to visualize the results as shown in Section 12.9. Results are discussed and explained in Section 12.10.

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

Section 12.11 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 12.3.
2. Specification of the preprocessing model, see Section 12.4. 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 12.5. 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 12.6.0.1. 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 12.7.5.
6. Calling SPOT with the corresponding parameters, see Section 12.8.4. The results are stored in a dictionary and are available for further analysis.
7. Presentation, visualization and interpretation of the results, see Section 12.10.

## 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

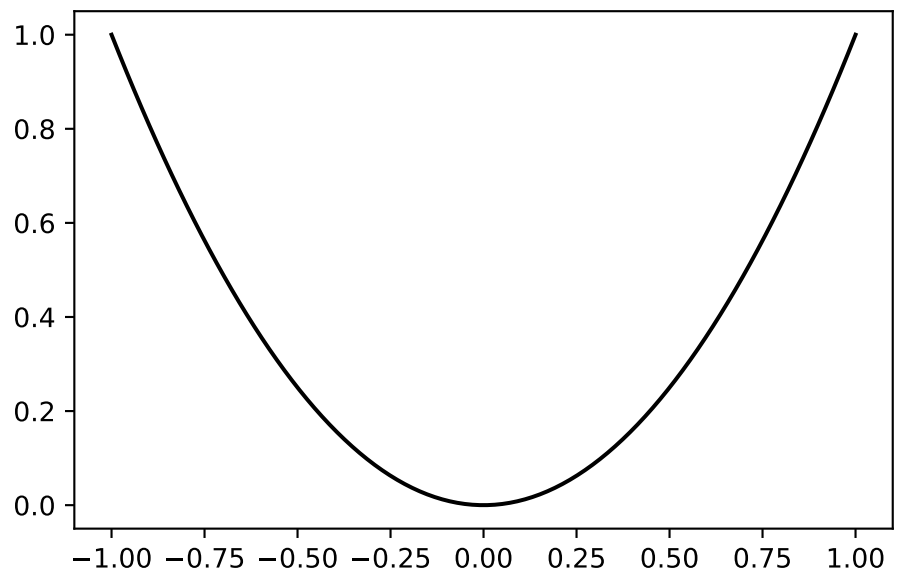
The `spotPython` package provides several classes of objective functions. We will use an analytical objective function, i.e., a function that can be described by a (closed) formula:

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

```
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()
```

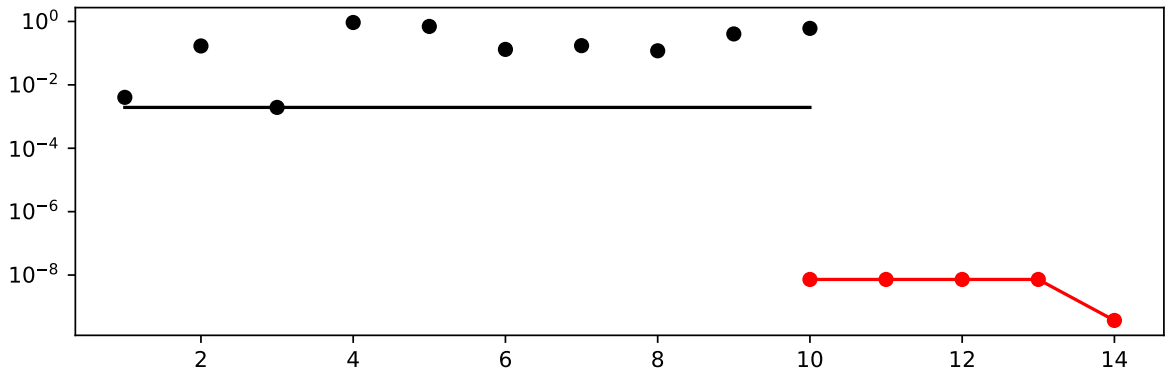
```
<spotPython.spot.spot.Spot at 0x162b20a90>
```

```
spot_0.print_results()
```

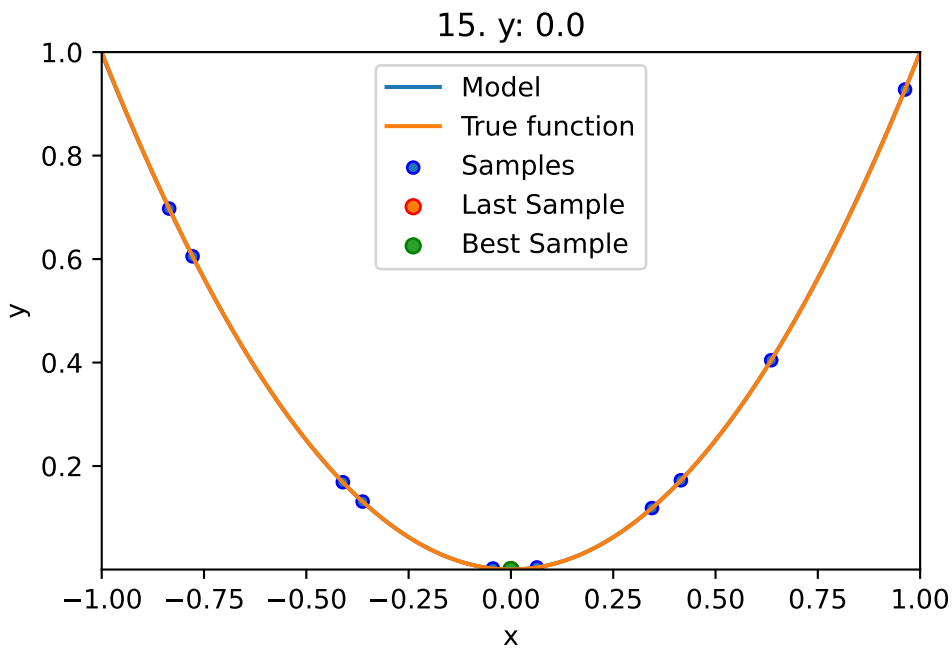
```
min y: 3.696886711914087e-10
x0: 1.922728975158508e-05
```

```
[['x0', 1.922728975158508e-05]]
```

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




```
spot_0.plot_model()
```




## 1.4 Spot Parameters: fun_evals, init_size and show_models

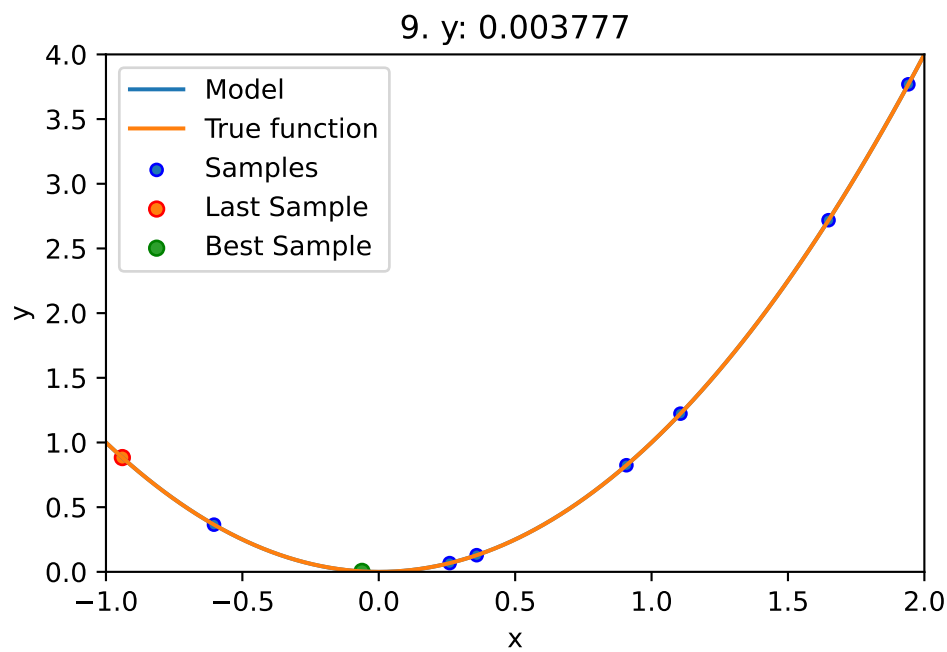
We will modify three parameters:

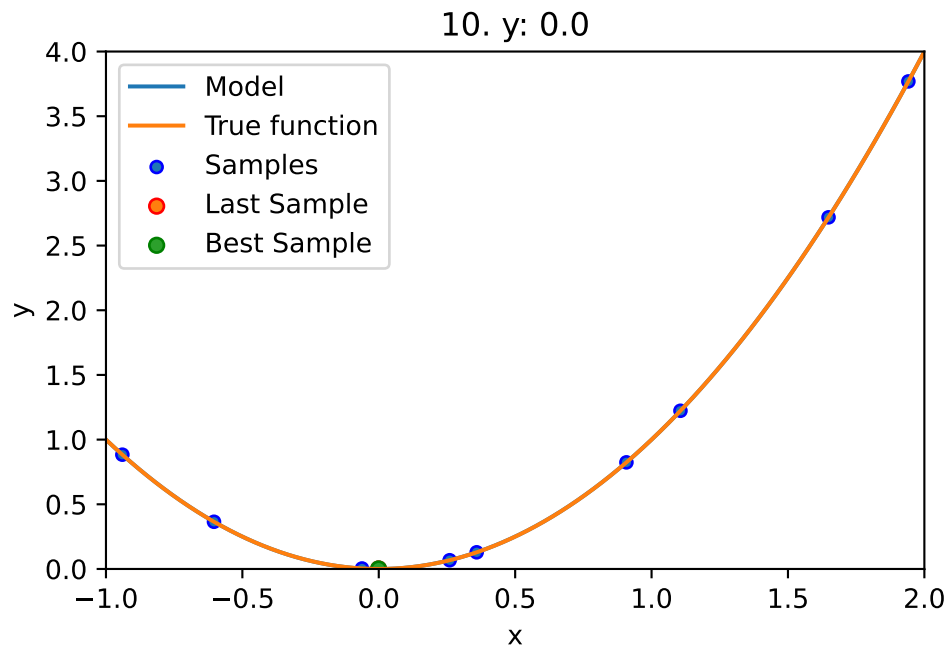
1. The number of function evaluations (`fun_evals`)
2. The size of the initial design (`init_size`)

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 0x1635c3fd0>
```

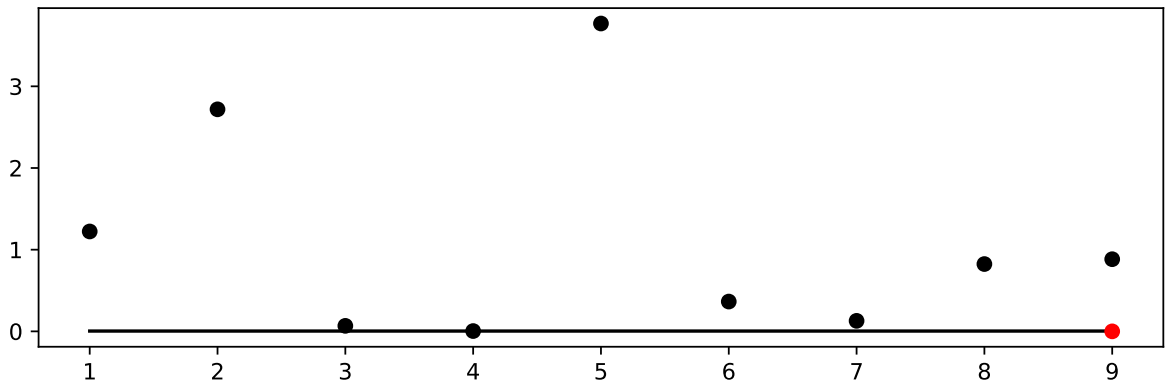
## 1.5 Print the Results

```
spot_1.print_results()
```

```
min y: 3.6779240309761575e-07
x0: -0.0006064589047063418

[['x0', -0.0006064589047063418]]
```

## 1.6 Show the Progress

```
spot_1.plot_progress()
```


3
2
1
0
1
2
3
4
5
6
7
8
9

# 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()
```

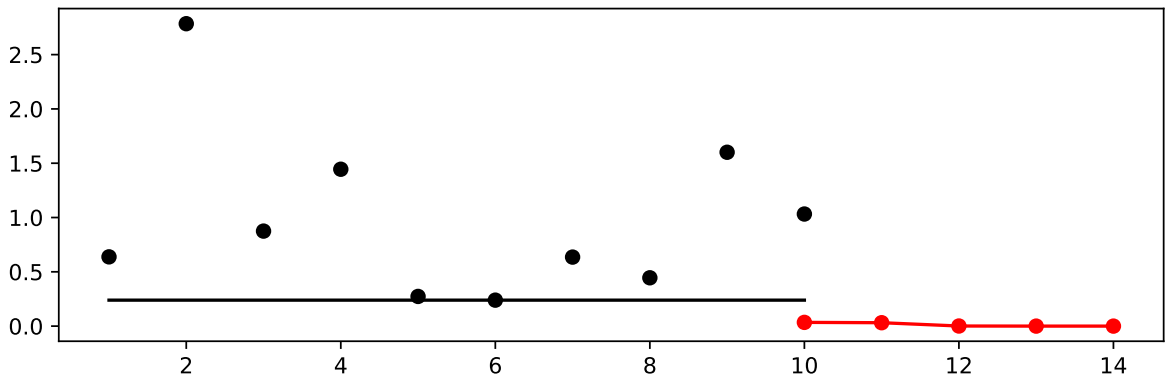
```
spotPython tuning: 0.03443344056467332 [#######---] 73.33%

spotPython tuning: 0.03134865993507926 [########--] 80.00%

spotPython tuning: 0.0009629342967936851 [#########-] 86.67%

spotPython tuning: 8.541951463966474e-05 [#########-] 93.33%

spotPython tuning: 6.285135731399678e-05 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x110782680>
```

### 2.1.2 Results

```
spot_3.print_results()
```

```
min y: 6.285135731399678e-05
Pressure: 0.005236109709736696
Temp: 0.0019572552655686714
Lambda: 0.005621713639718905

[['Pressure', 0.005236109709736696],
 ['Temp', 0.0019572552655686714],
 ['Lambda', 0.005621713639718905]]
```

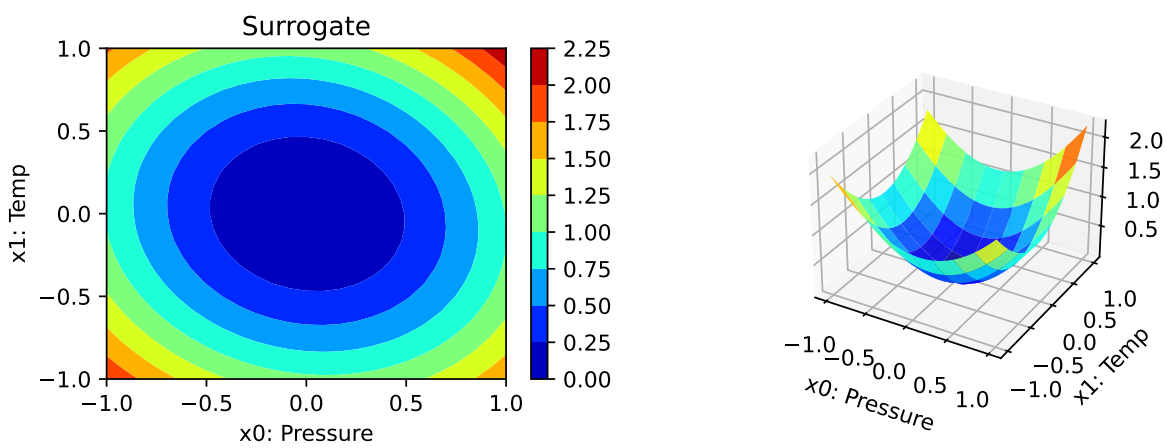
```
spot_3.plot_progress()
```




### 2.1.3 A Contour Plot

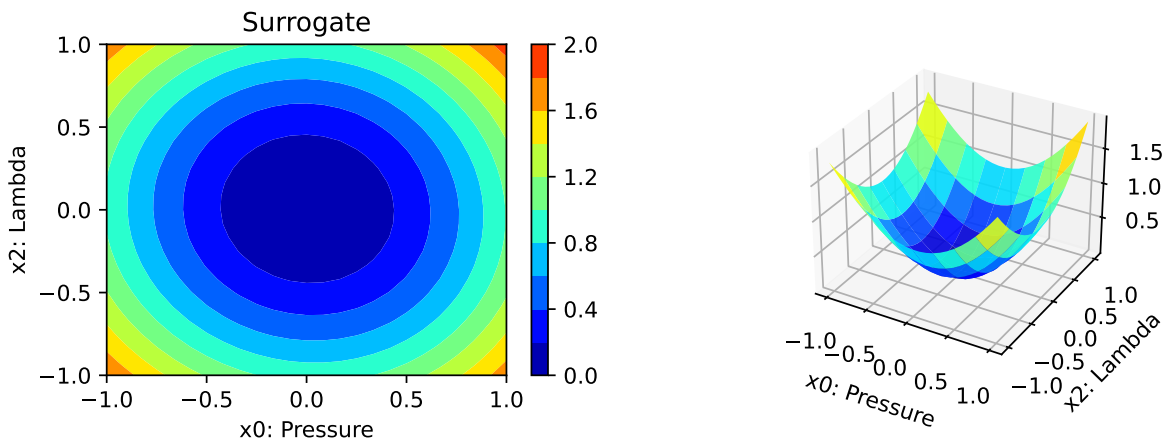
- We can select two dimensions, say $i = 0$ and $j = 1$, and generate a contour plot as follows.
  - Note: We have specified identical `min_z` and `max_z` values to generate comparable plots!

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



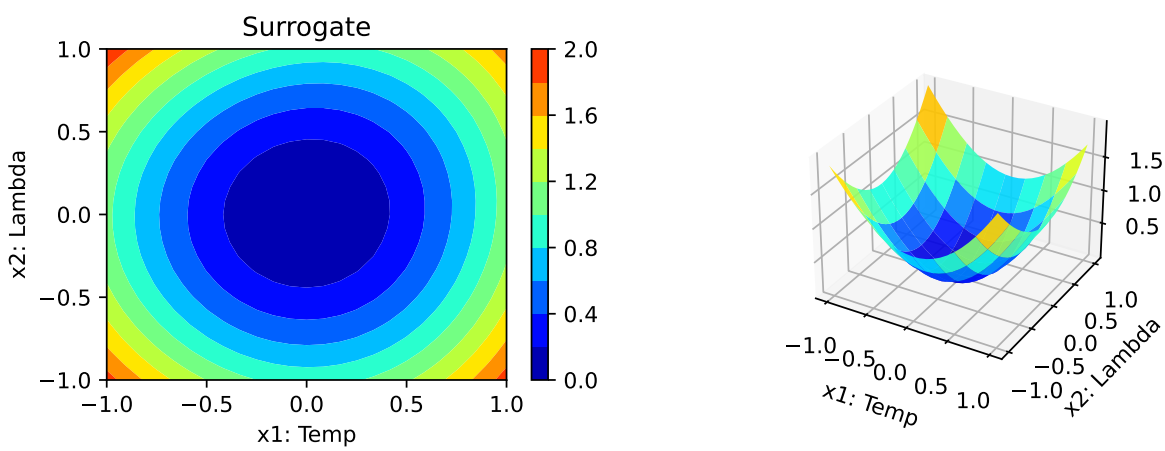

- In a similar manner, we can plot dimension $i = 0$ and $j = 2$:

```
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:  99.35185545837122
Temp:  99.99999999999999
```

```
Lambda:  94.31627052007231
```

```
[['Pressure', 99.35185545837122],
 ['Temp', 99.99999999999999],
 ['Lambda', 94.31627052007231]]
```

## 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 addtional 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()
```

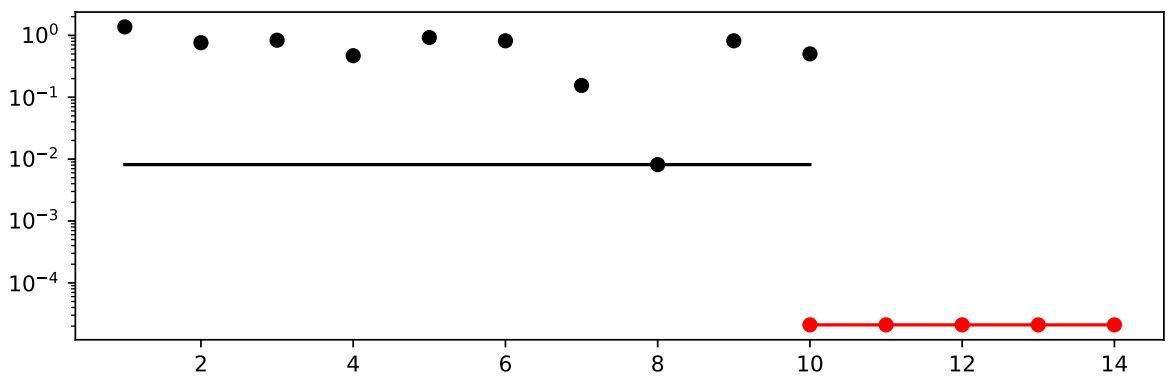
```
<spotPython.spot.spot.Spot at 0x13f45cf40>
```

### 3.1.2 Results

```
spot_2.print_results()
```

```
min y: 2.093282610941807e-05
x0: 0.0016055267473267492
x1: 0.00428428640184529

[['x0', 0.0016055267473267492], ['x1', 0.00428428640184529]]
```

```
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 0x16a0f5ff0>
```

### 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([0.19447342, 0.30813872])
```

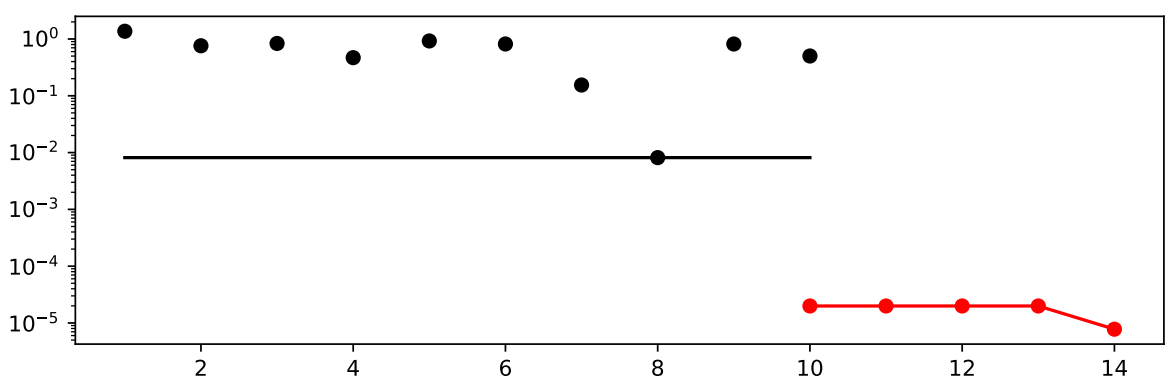
- Since the surrogate from the isotropic setting was stored as `spot_2`, we can also take a look at the `theta` value from this model:

```
spot_2.surrogate.theta
```

```
array([0.26287447])
```

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

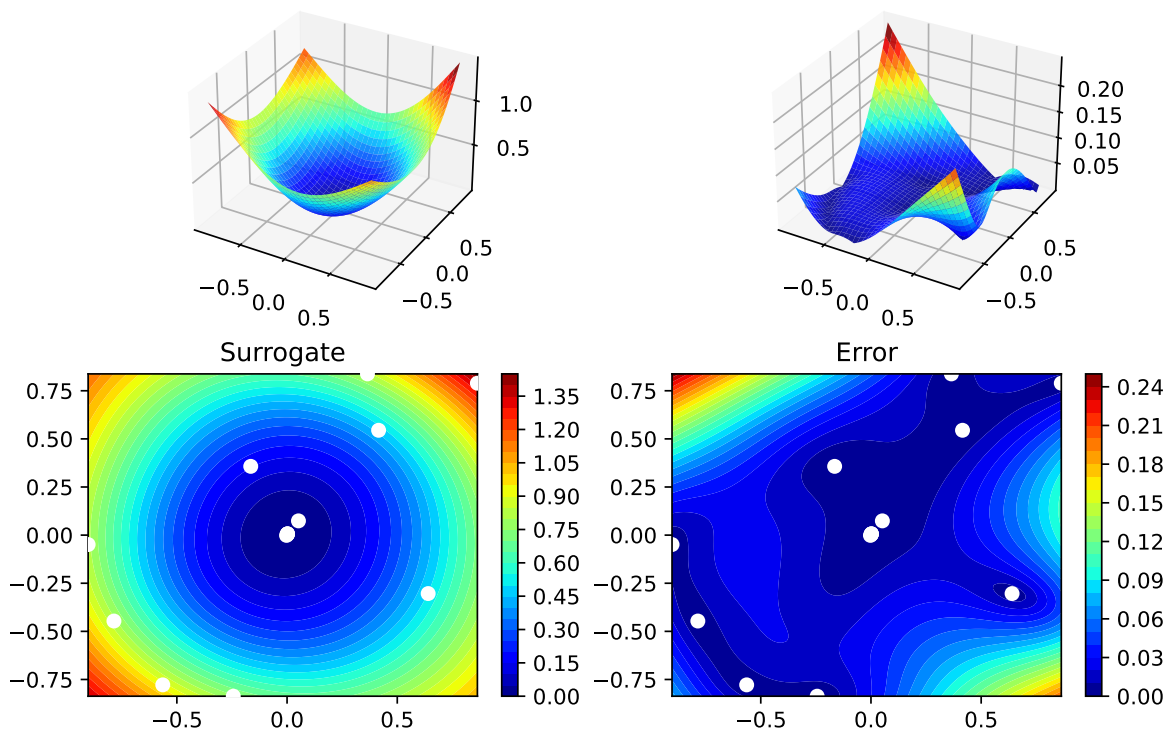
```
spot_2_anisotropic.plot_progress(log_y=True)
```



```
spot_2_anisotropic.print_results()
```

```
min y: 7.77061191821505e-06
x0: -0.0024488252797500764
x1: -0.0013318658594137815

[['x0', -0.0024488252797500764], ['x1', -0.0013318658594137815]]
```

```
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 * (x2 - b * x1**2 + c * x1 - r) ** 2 + s * (1 - t) * np.cos(x1) + 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 0x17fba8940>
```

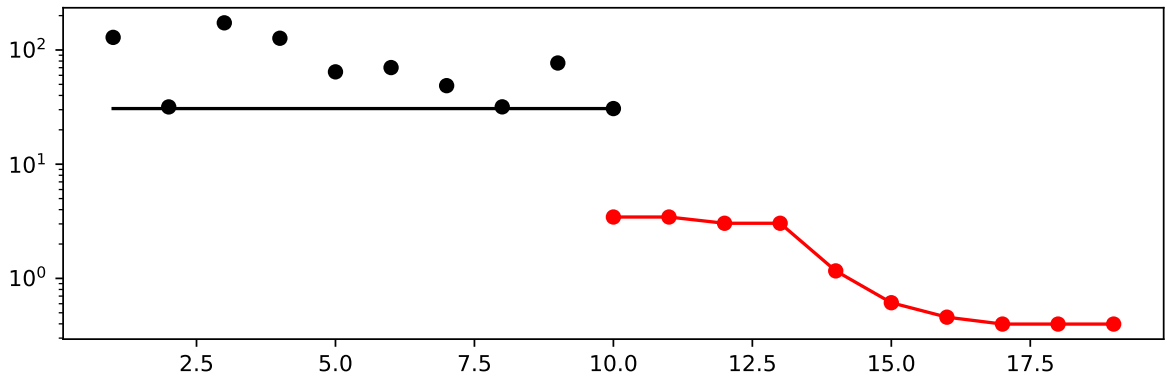
### 4.1.3 Print the Results

```
spot_2.print_results()
```

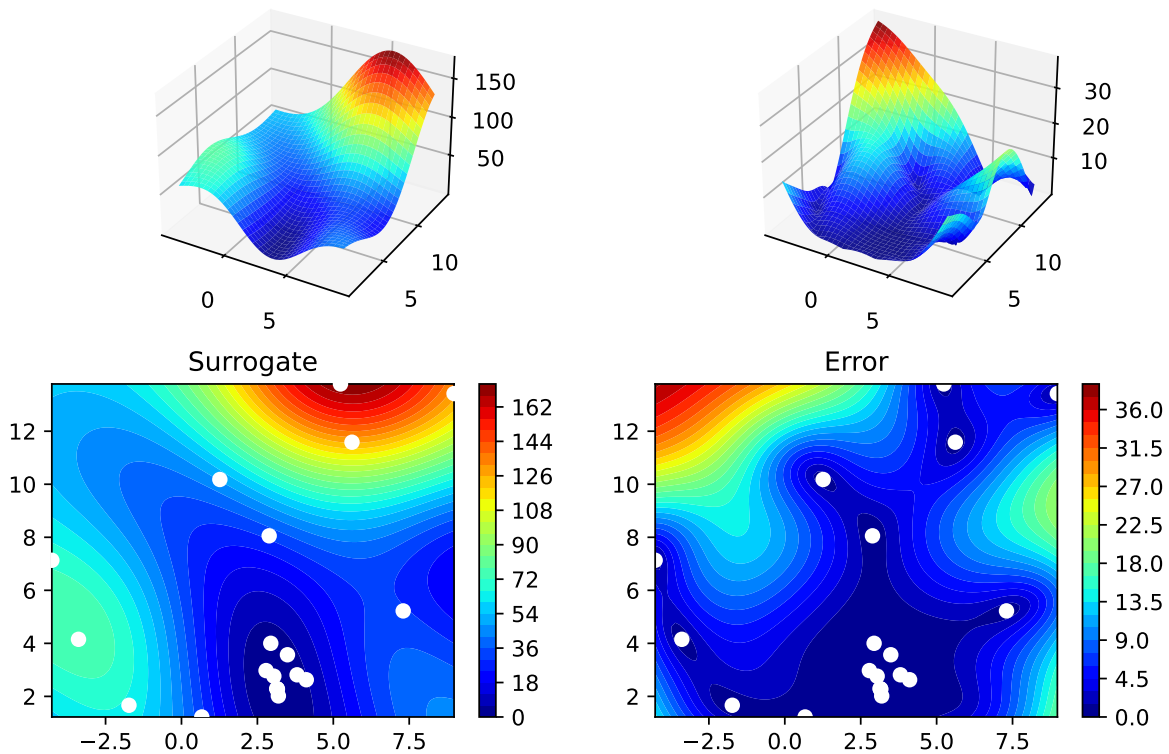
```
min y: 0.3982295132785083
x0: 3.135528626303215
x1: 2.2926027772585886

[['x0', 3.135528626303215], ['x1', 2.2926027772585886]]
```

### 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 explicitely 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 surogate model with the command `isinstance`:

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

```
True
```

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

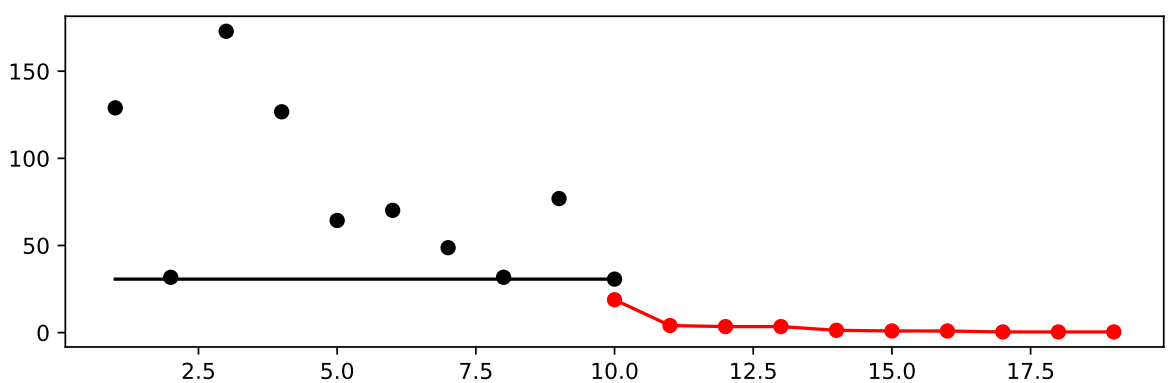
```
True
```

- 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 0x28ec27c10>
```

```
spot_2_GP.plot_progress()
```




```
spot_2_GP.print_results()
```

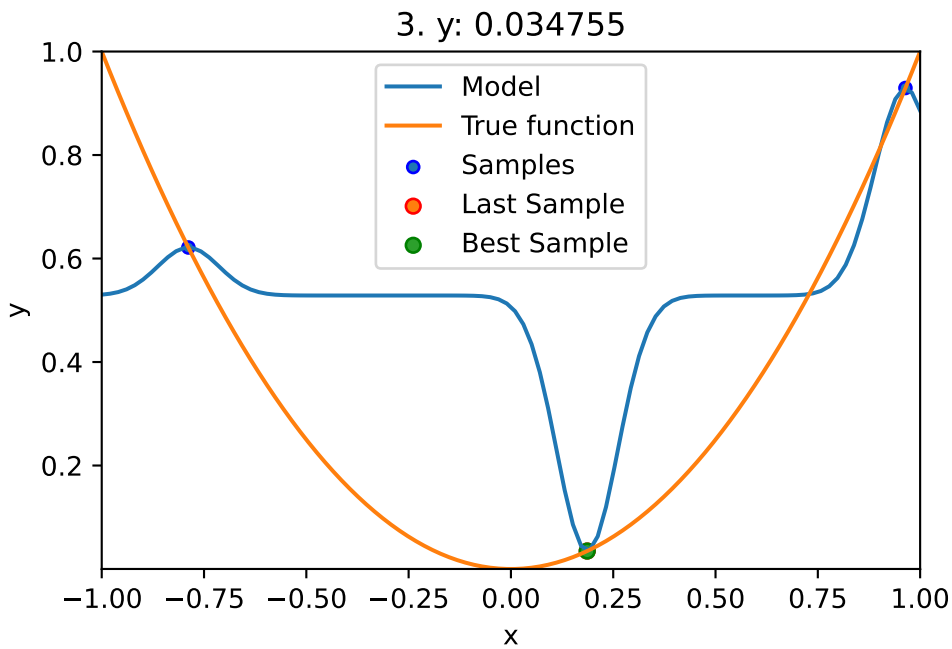
```
min y: 0.3979824609688851
x0: 3.1459832065501216
x1: 2.2731764882537484

[['x0', 3.1459832065501216], ['x1', 2.2731764882537484]]
```

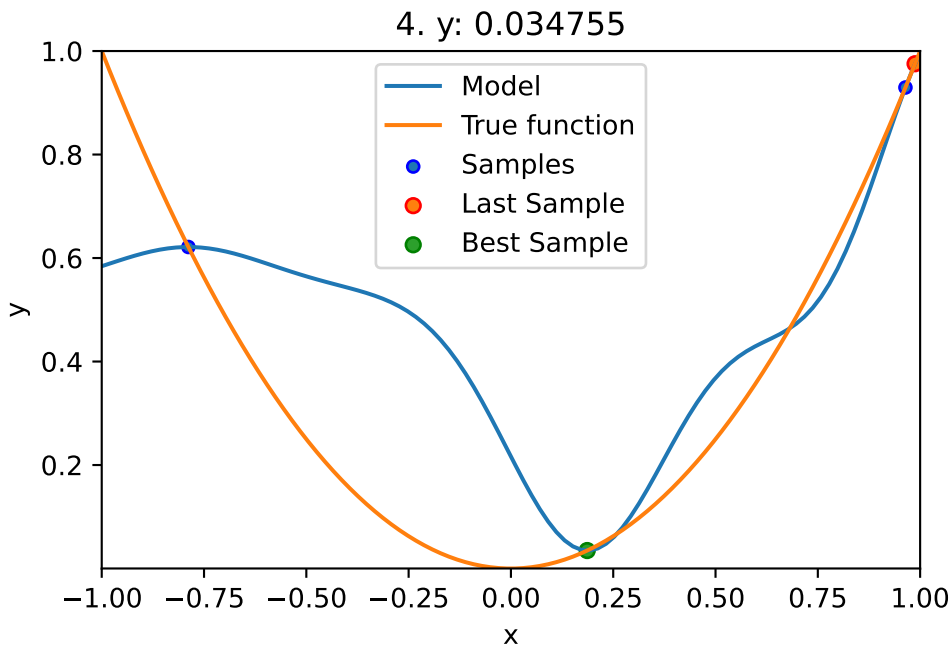
## 4.3 Example: One-dimensional Sphere Function With `spotPython`'s Kriging

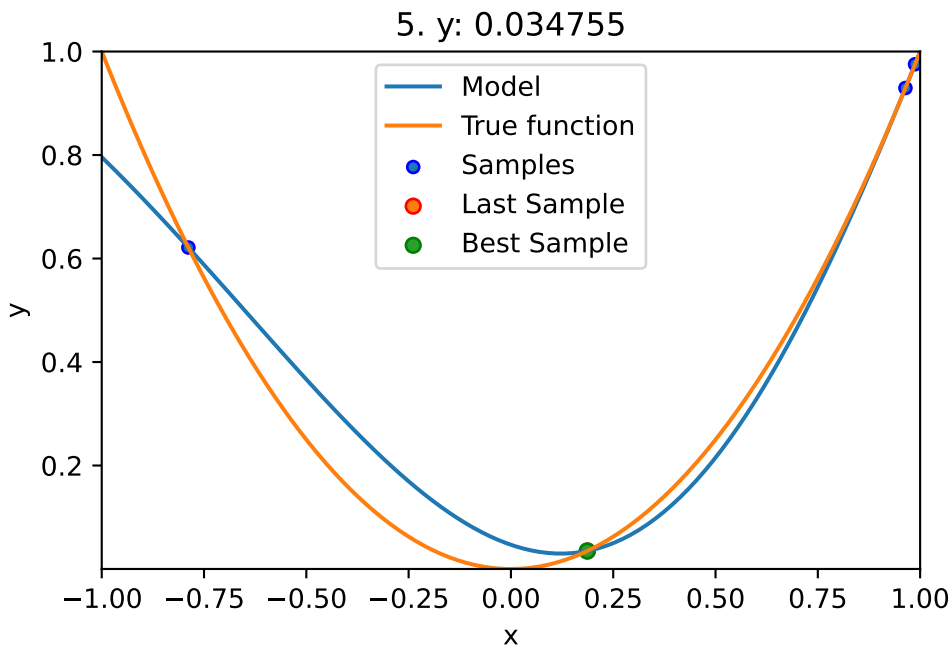
- In this example, we will use an one-dimensional function, which allows us to visualize the optimization process.
  - `show_models= True` is added to the argument list.

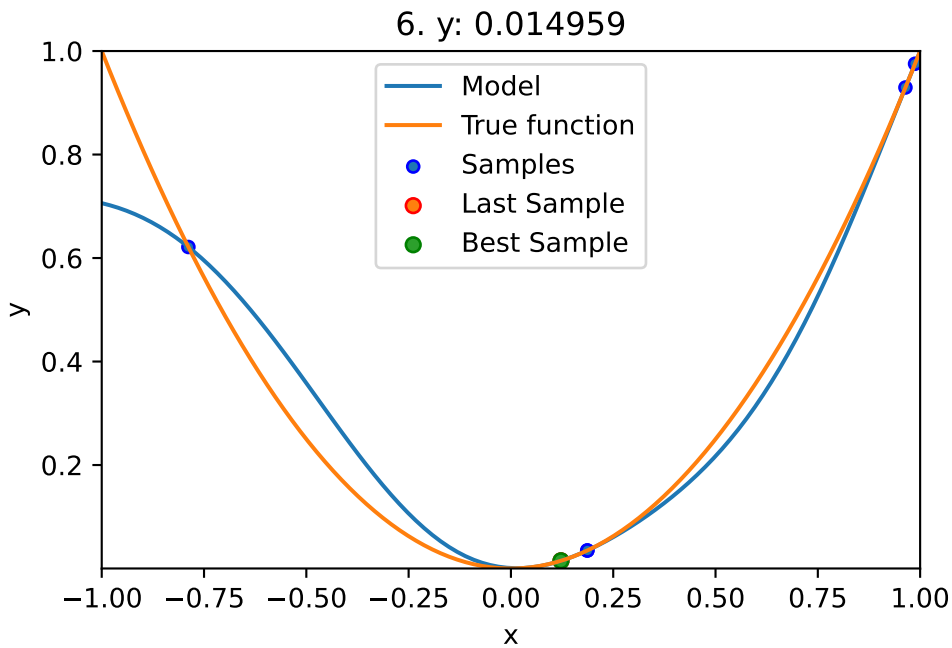
```
from spotPython.fun.objectivefunctions import analytical
lower = np.array([-1])
upper = np.array([1])
fun = analytical(seed=123).fun_sphere
```

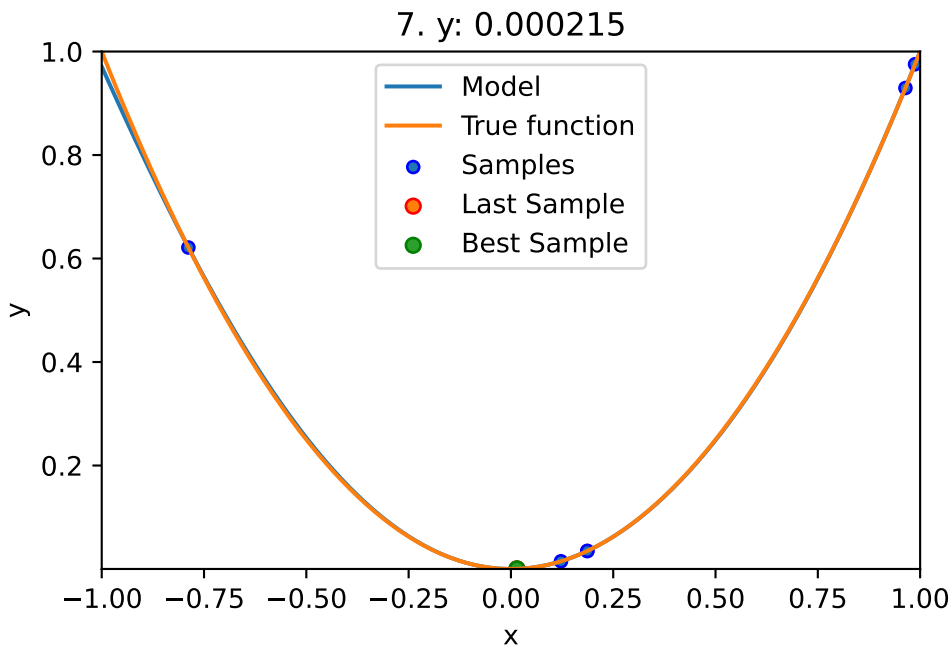
```
spot_1 = spot.Spot(fun=fun,
                   lower = lower,
                   upper = upper,
                   fun_evals = 10,
                   max_time = inf,
                   seed=123,
                   show_models= True,
                   tolerance_x = np.sqrt(np.spacing(1)),
      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

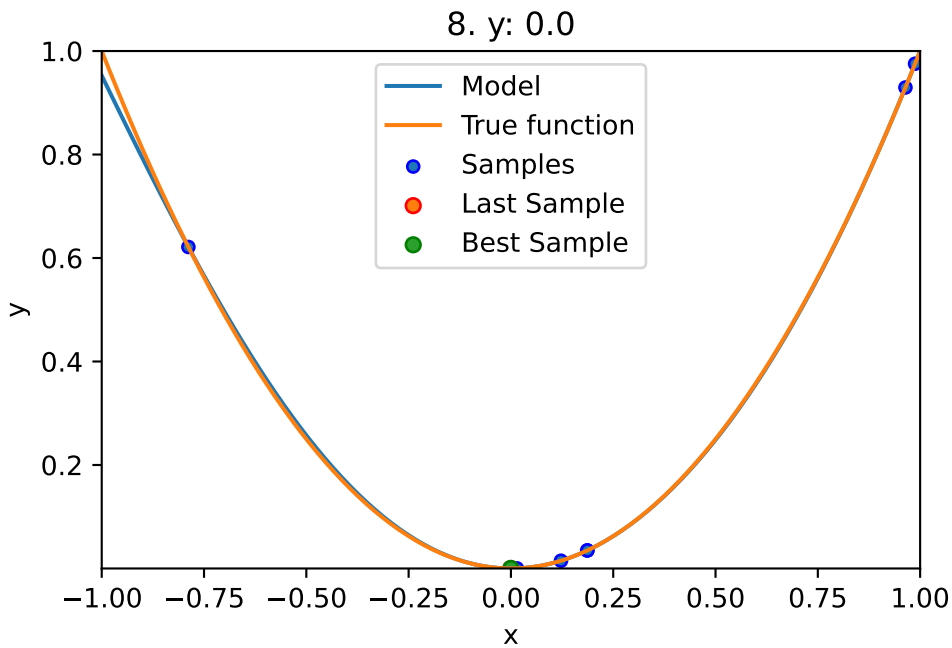

3. y: 0.034755
Model
True function
Samples
Last Sample
Best Sample
y
x
1.0
0.8
0.6
0.4
0.2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

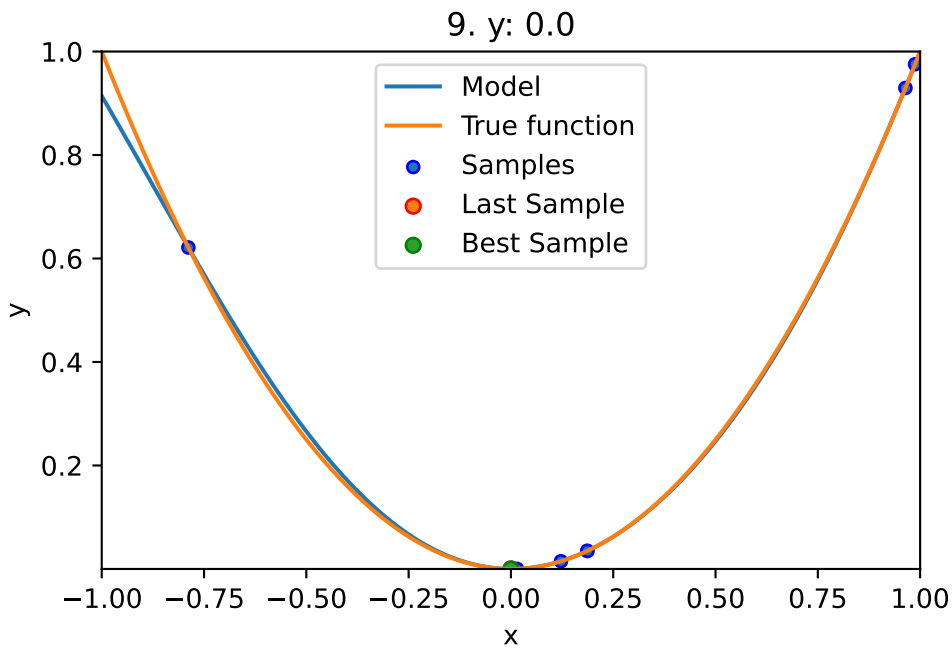

4. y: 0.034755
Model
True function
Samples
Last Sample
Best Sample
y
x
1.0
0.8
0.6
0.4
0.2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

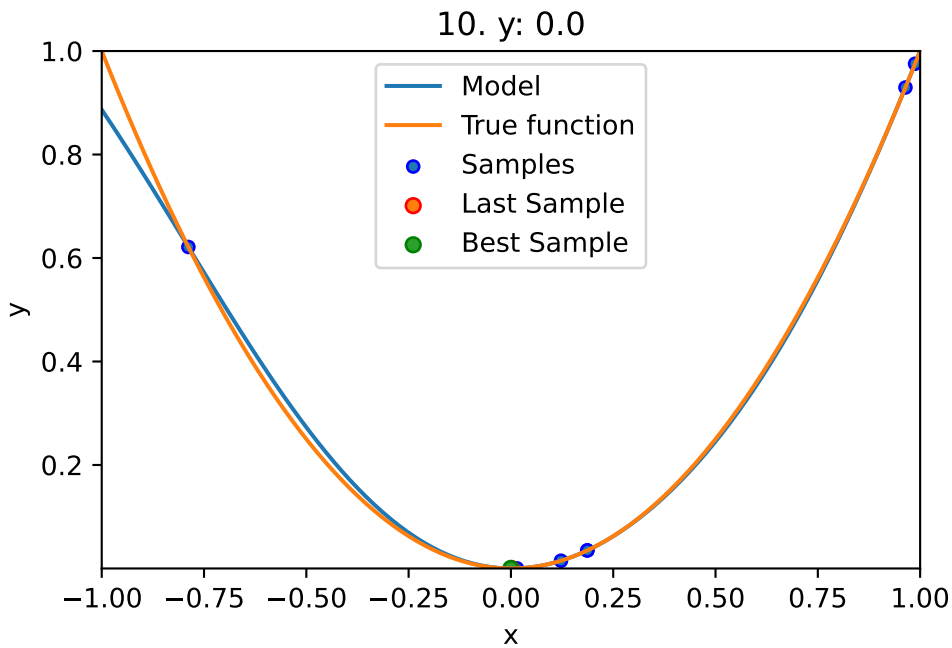

5. y: 0.034755
Model
True function
Samples
Last Sample
Best Sample
y
x
1.0
0.8
0.6
0.4
0.2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


6. y: 0.014959
Model
True function
Samples
Last Sample
Best Sample
y
x
1.0
0.8
0.6
0.4
0.2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


7. y: 0.000215
Model
True function
Samples
Last Sample
Best Sample
y
x
1.0
0.8
0.6
0.4
0.2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


8. y: 0.0
Model
True function
Samples
Last Sample
Best Sample
y
x
1.0
0.8
0.6
0.4
0.2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00



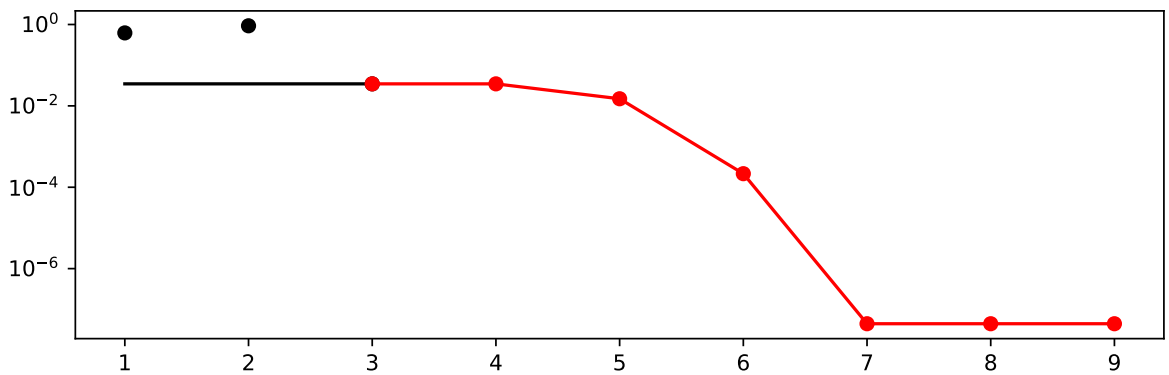

```
<spotPython.spot.spot.Spot at 0x28f0039a0>
```

### 4.3.1 Results

```
spot_1.print_results()
```

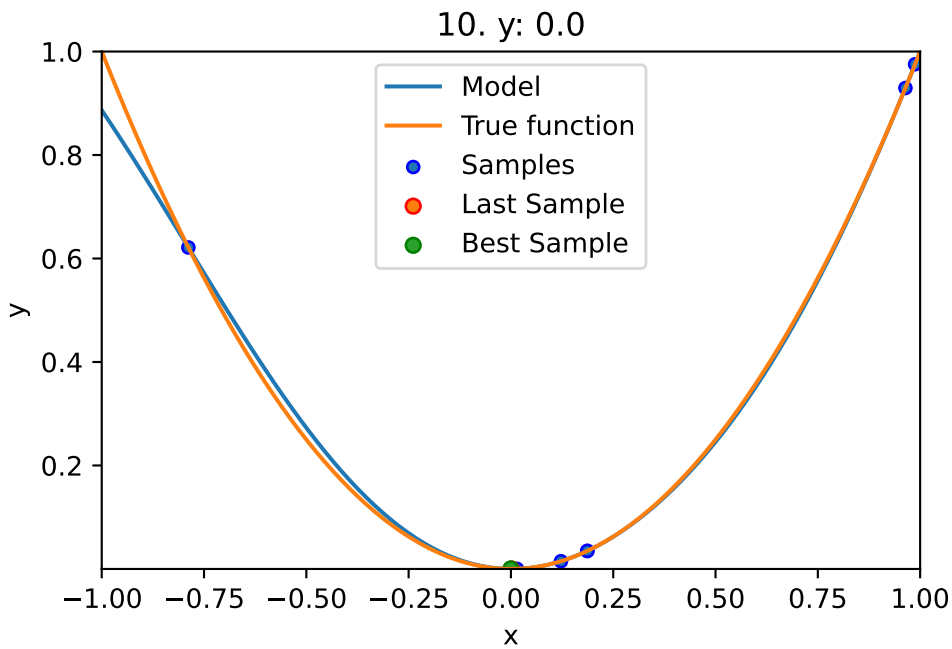
```
min y: 4.41925228274096e-08
x0: -0.00021022017702259125

[['x0', -0.00021022017702259125]]
```

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




- The method `plot_model` plots the final surrogate:

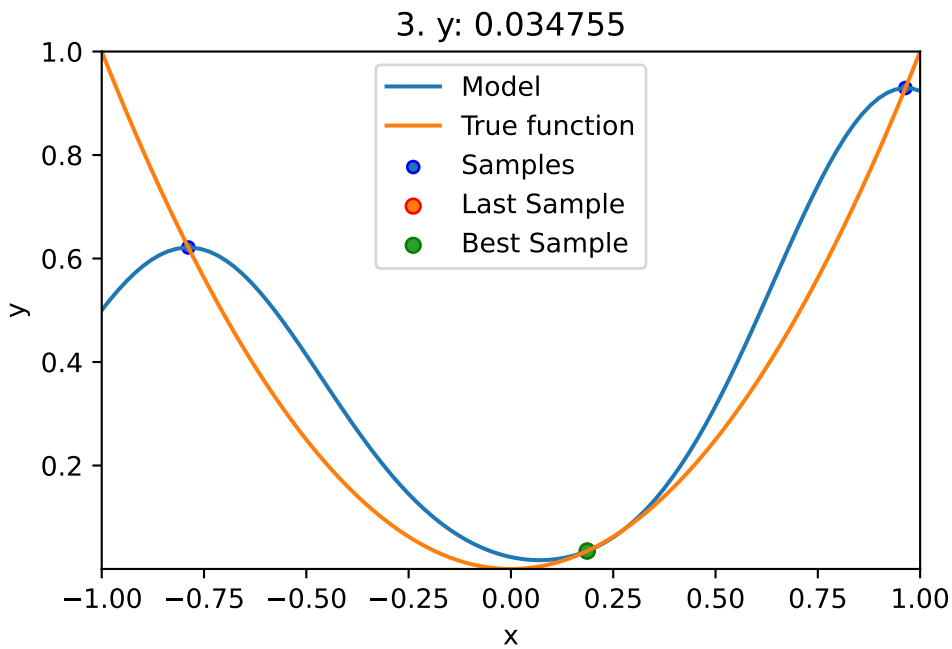
```
spot_1.plot_model()
```

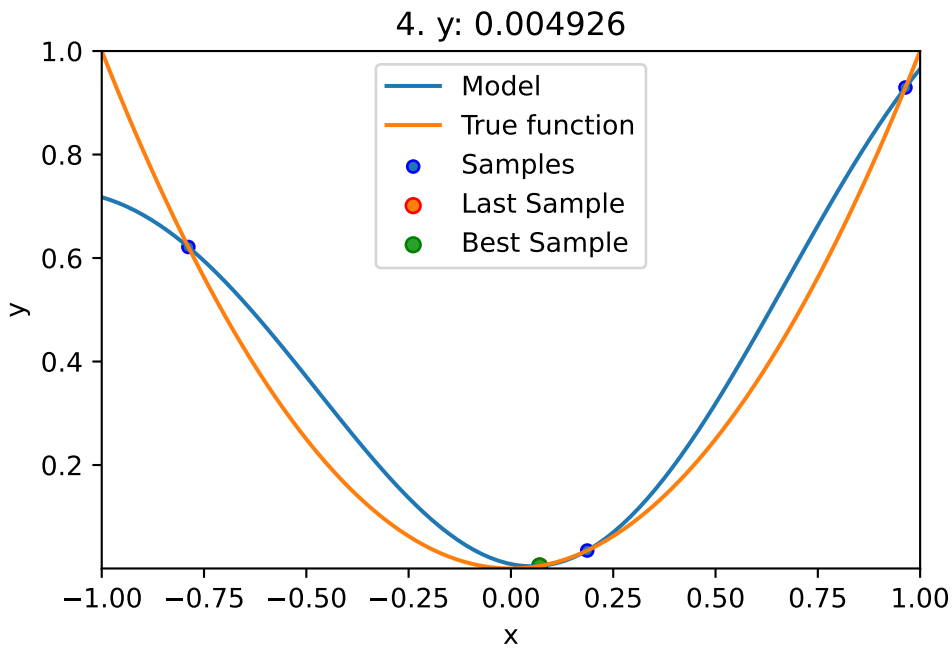


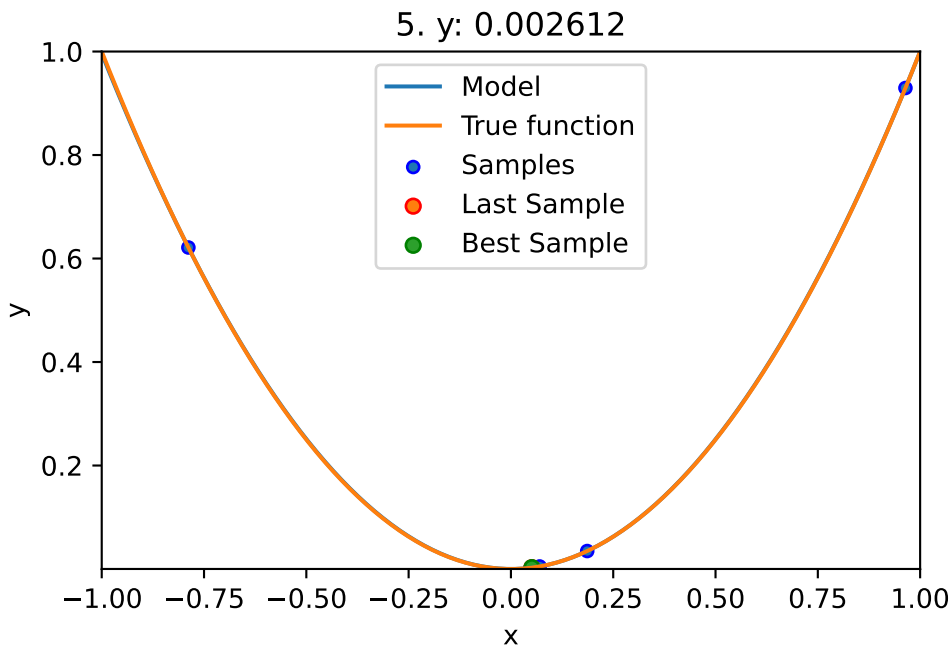

## 4.4 Example: `Sklearn` Model GaussianProcess

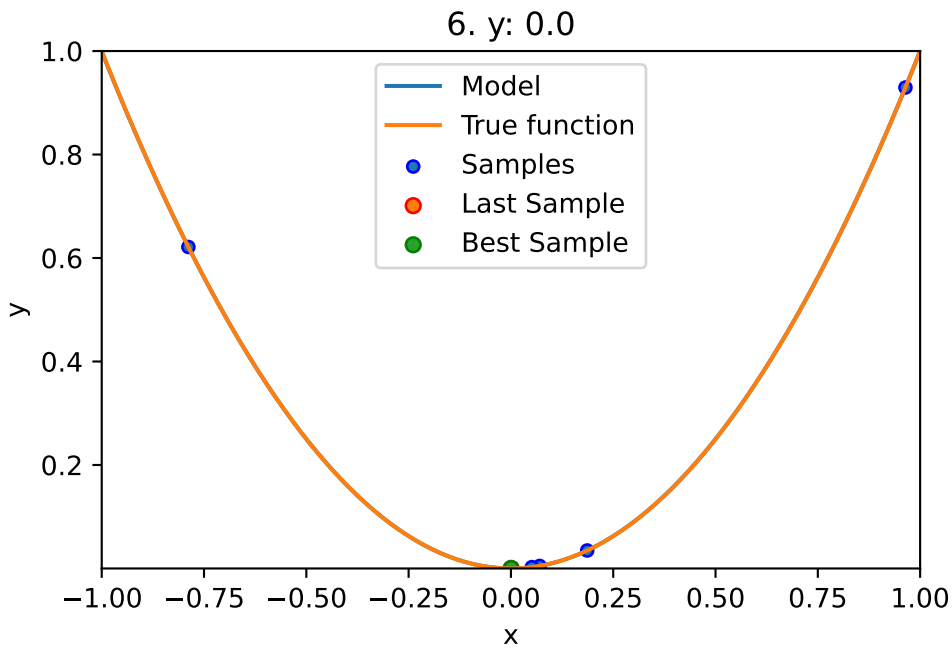
- This example visualizes the search process on the `GaussianProcessRegression` surrogate from `sklearn`.
- Therefore `surrogate = S_GP` is added to the argument list.

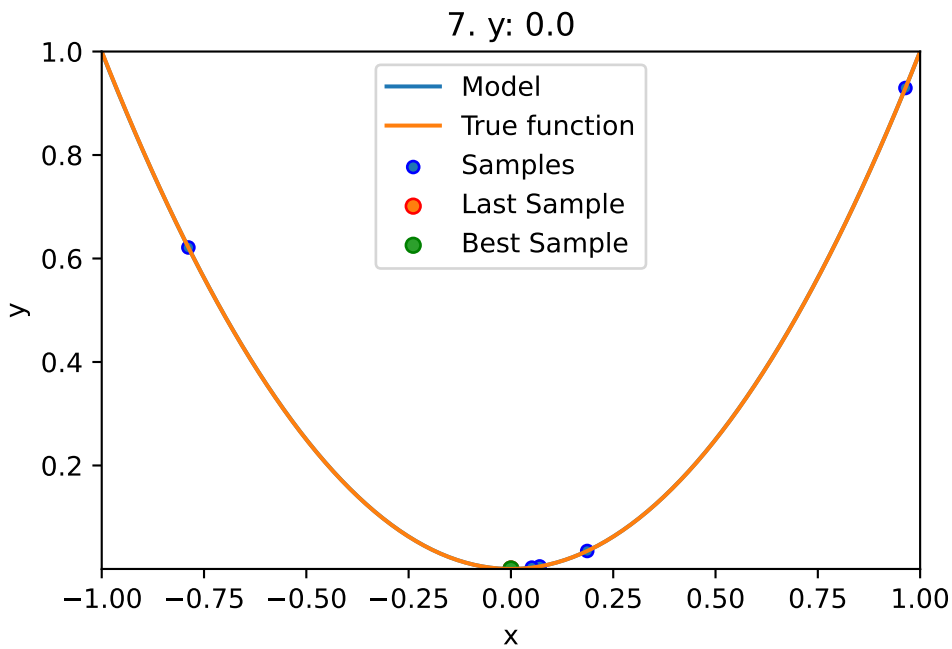
```
fun = analytical(seed=123).fun_sphere
spot_1_GP = spot.Spot(fun=fun,
                    lower = lower,
                    upper = upper,
                    fun_evals = 10,
                    max_time = inf,
                    seed=123,
                    show_models= True,
                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

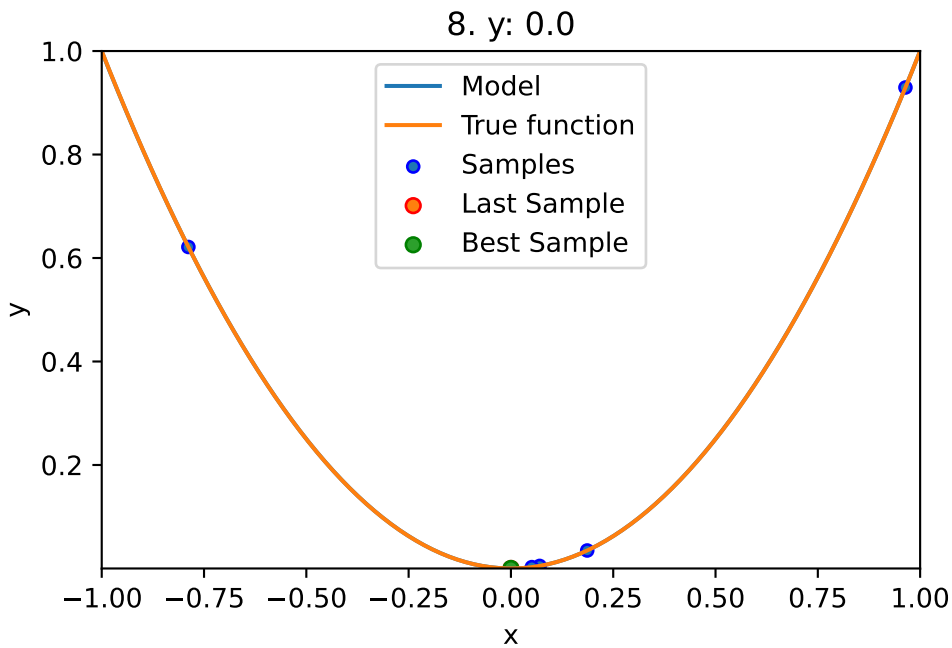

3. y: 0.034755
Model
True function
Samples
Last Sample
Best Sample
y
x
1.0
0.8
0.6
0.4
0.2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

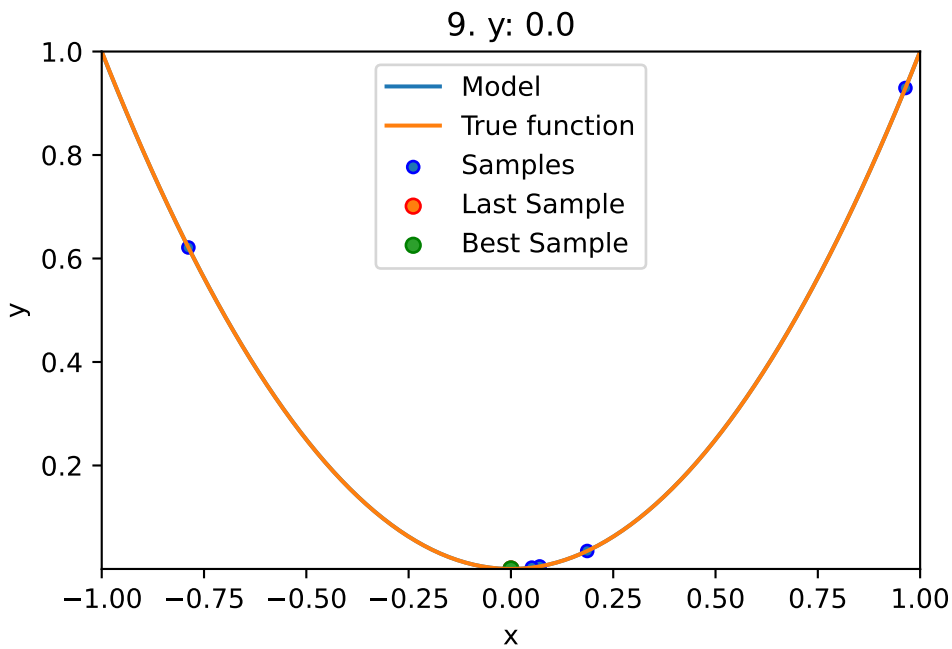

4. y: 0.004926
Model
True function
Samples
Last Sample
Best Sample
y
x
1.0
0.8
0.6
0.4
0.2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

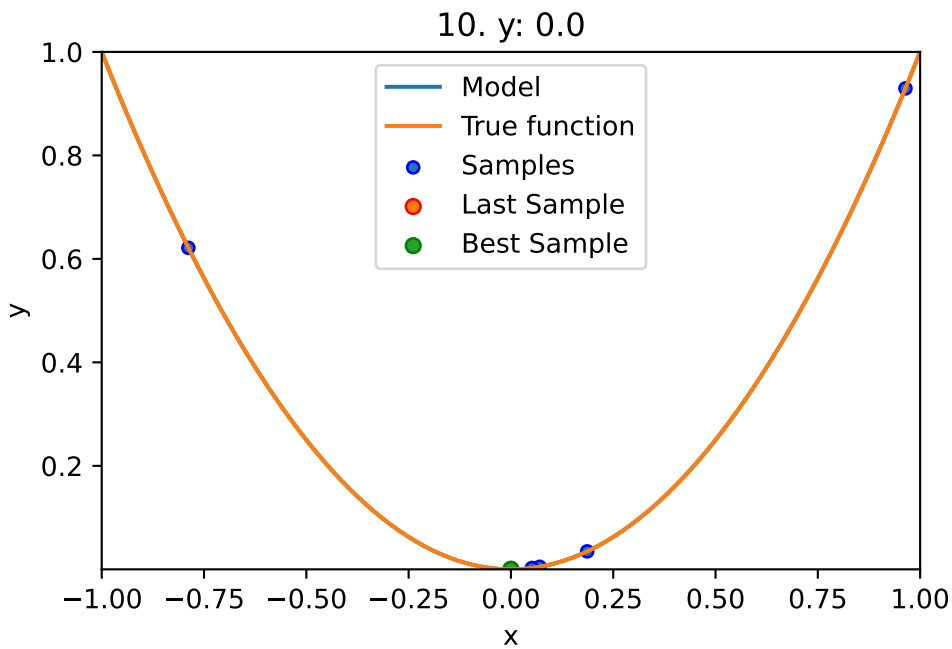

5. y: 0.002612
Model
True function
Samples
Last Sample
Best Sample
y
x
1.0
0.8
0.6
0.4
0.2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


6. y: 0.0
Model
True function
Samples
Last Sample
Best Sample
y
x
1.0
0.8
0.6
0.4
0.2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


7. y: 0.0
Model
True function
Samples
Last Sample
Best Sample
y
x
1.0
0.8
0.6
0.4
0.2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


8. y: 0.0
Model
True function
Samples
Last Sample
Best Sample
y
x
1.0
0.8
0.6
0.4
0.2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00



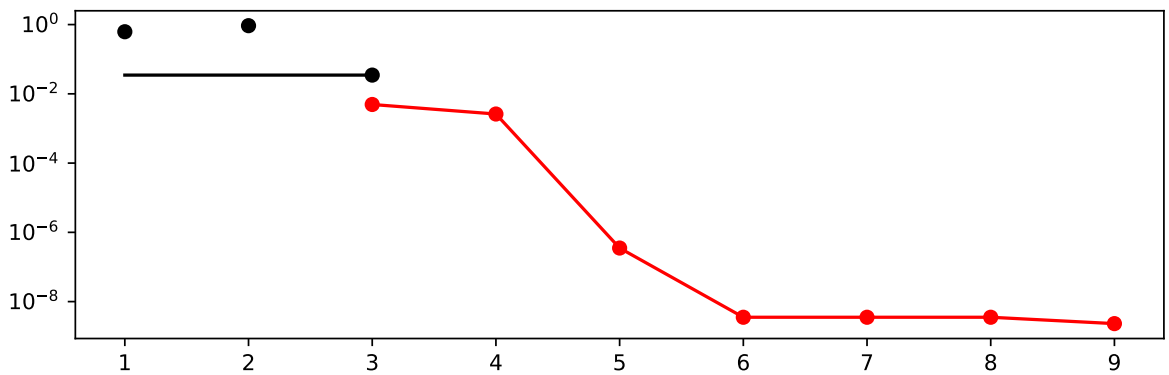




```
<spotPython.spot.spot.Spot at 0x2b8764820>
```

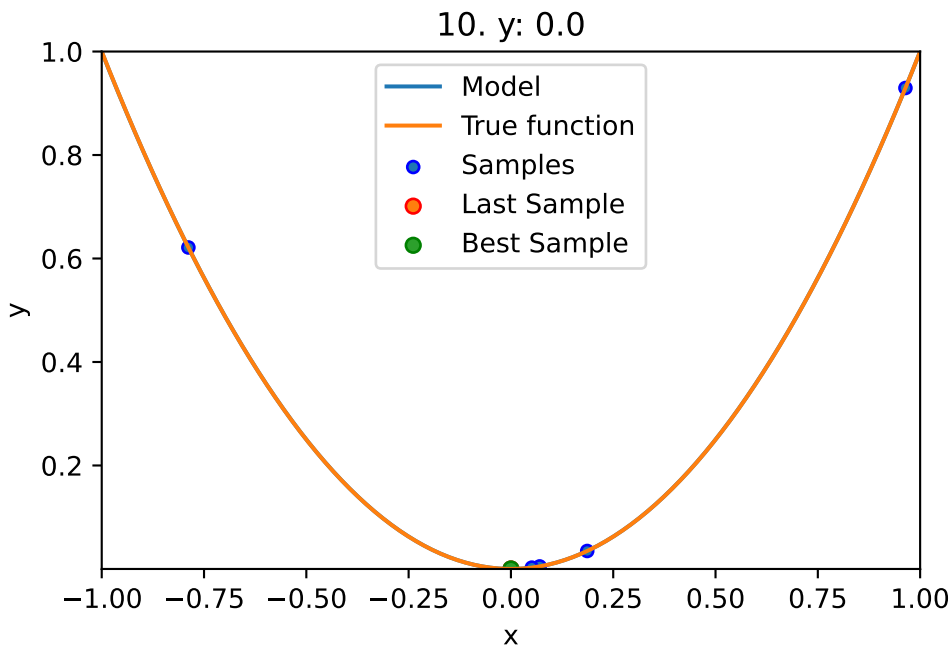
```
spot_1_GP.print_results()
```

```
min y: 2.3124131929966138e-09
x0: 4.8087557569465034e-05

[['x0', 4.8087557569465034e-05]]
```

```
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 form 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

  $$y = a * (x2 - b * x1 * *2 + c * x1 - r) * *2 + s * (1 - t) * cos(x1) + s,$$

  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 $(9.42478, 2.475)$.

- Input Domain: This function is usually evaluated on the square x1 in [-5, 10] x x2 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 Evalution 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 optimiers 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 0x106981690>
```

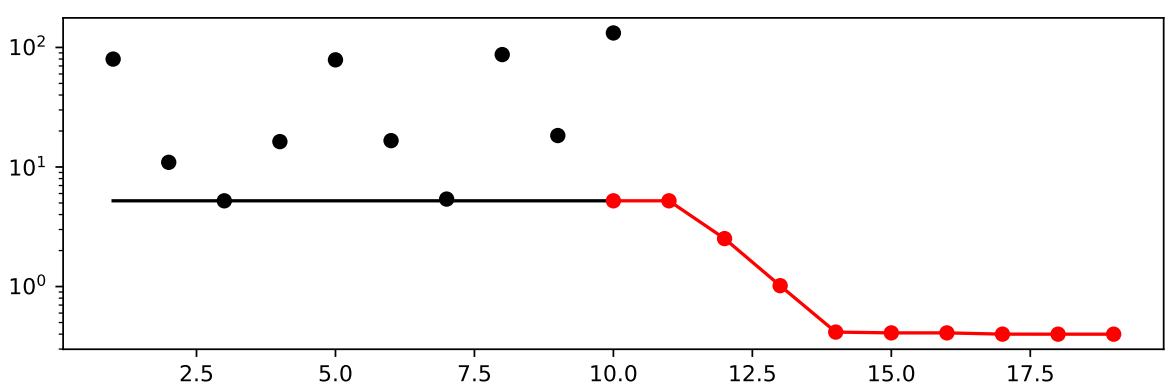
## 5.3 Print the Results

```
spot_de.print_results()
```

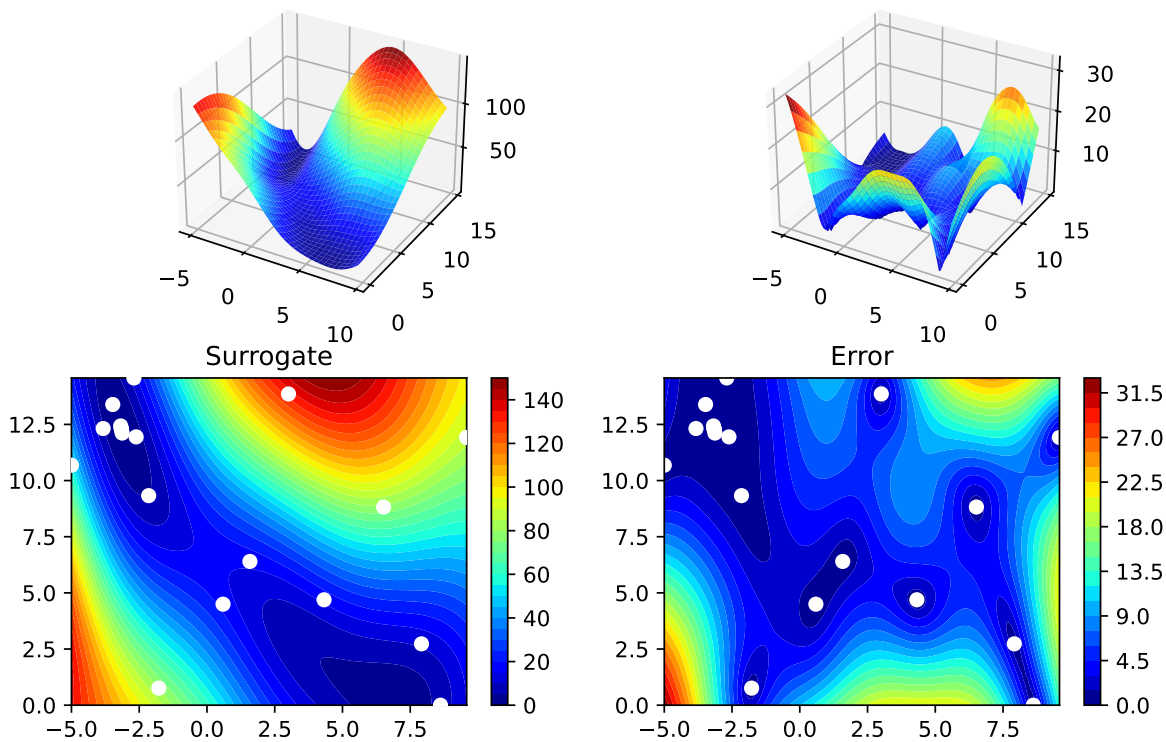
```
min y: 0.39951958110619046
x0: -3.1570201165683587
x1: 12.289980569430284

[['x0', -3.1570201165683587], ['x1', 12.289980569430284]]
```

## 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/pl
- 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]
```

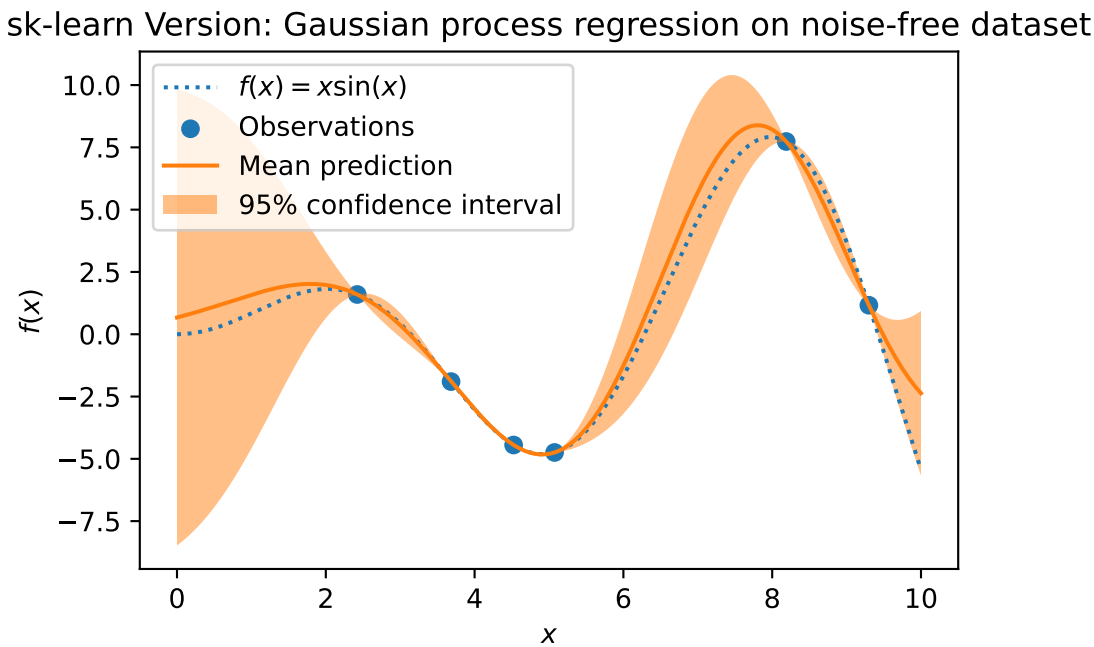
### 6.1.2 Building the Surrogate With `Sklearn`

- The model building with `sklearn` consisits 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 `Sklearn`Model

```
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")
```



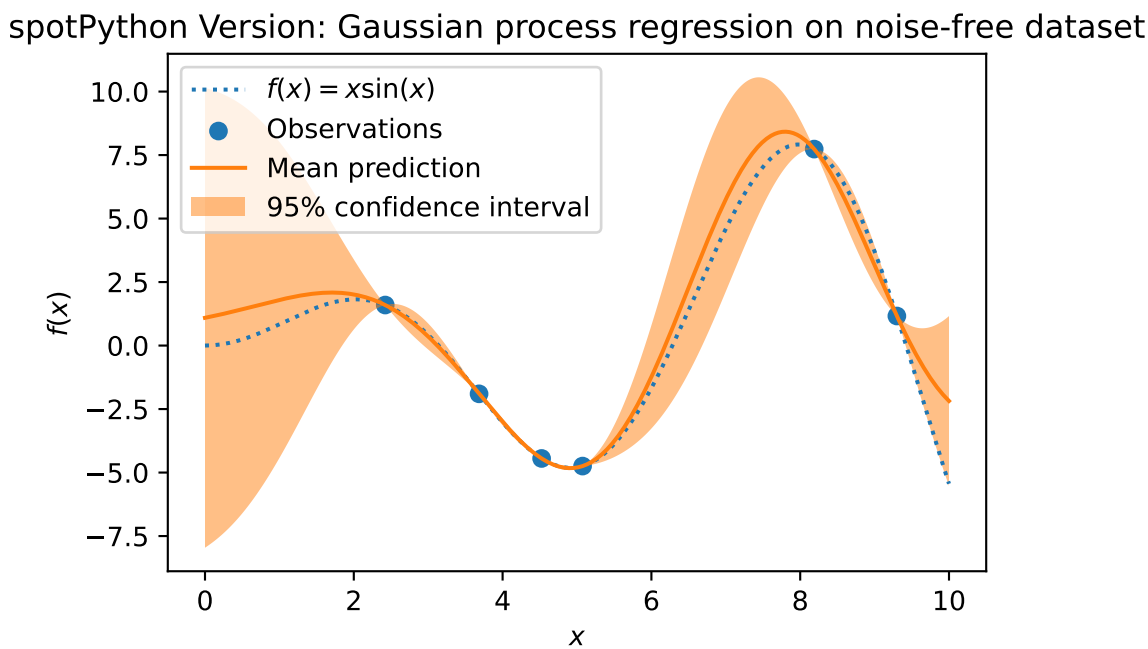

### 6.1.4 The `spotPython` Version

- 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")
```

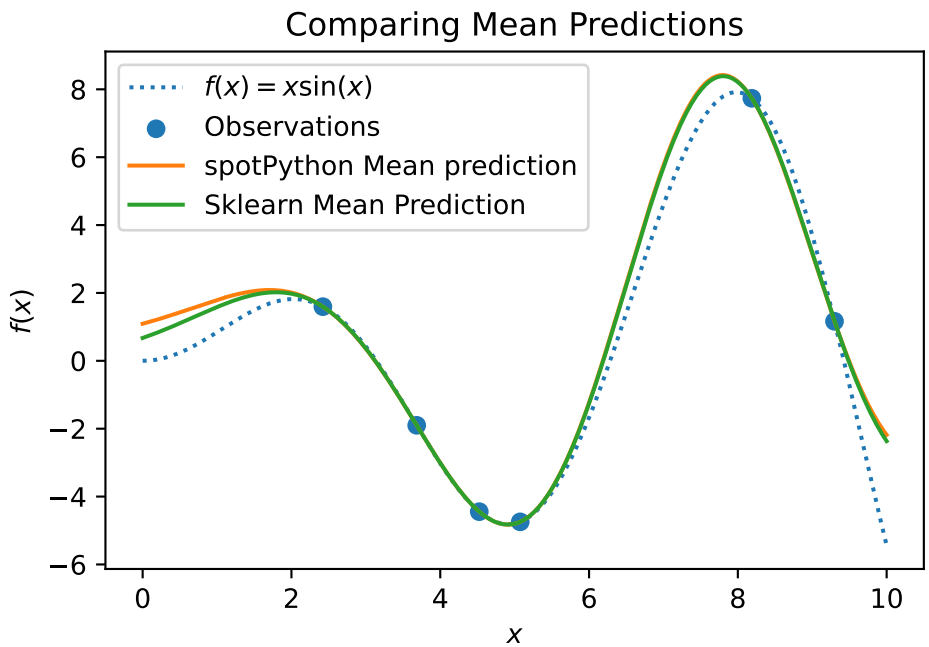
```
plt.plot(X, y, label=r"$f(x) = x \sin(x)$", linestyle="dotted")
plt.scatter(X_train, y_train, label="Observations")
plt.plot(X, S_mean_prediction, label="Mean prediction")
plt.fill_between(
    X.ravel(),
    S_mean_prediction - 1.96 * S_std_prediction,
    S_mean_prediction + 1.96 * S_std_prediction,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




### 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()
```

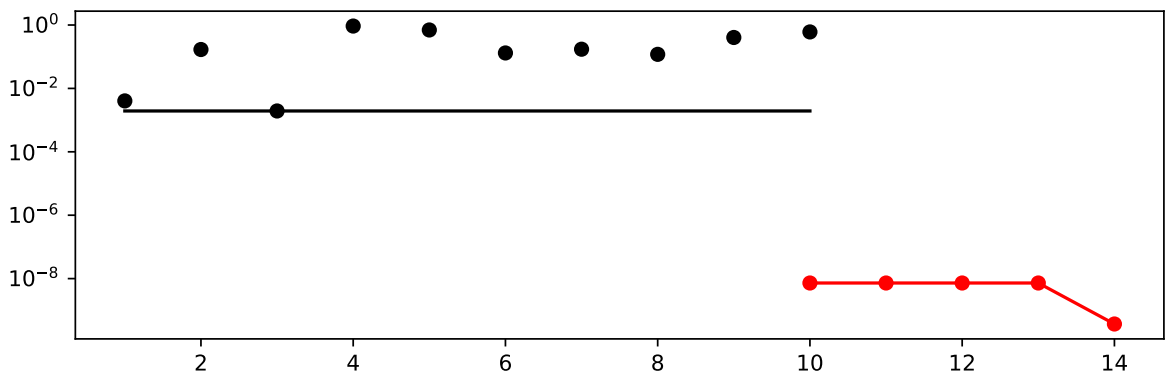
```
<spotPython.spot.spot.Spot at 0x14f5b8bb0>
```

### 7.1.2 Results

```
spot_1.print_results()
```

```
min y: 3.696886711914087e-10
x0: 1.922728975158508e-05

[['x0', 1.922728975158508e-05]]
```

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

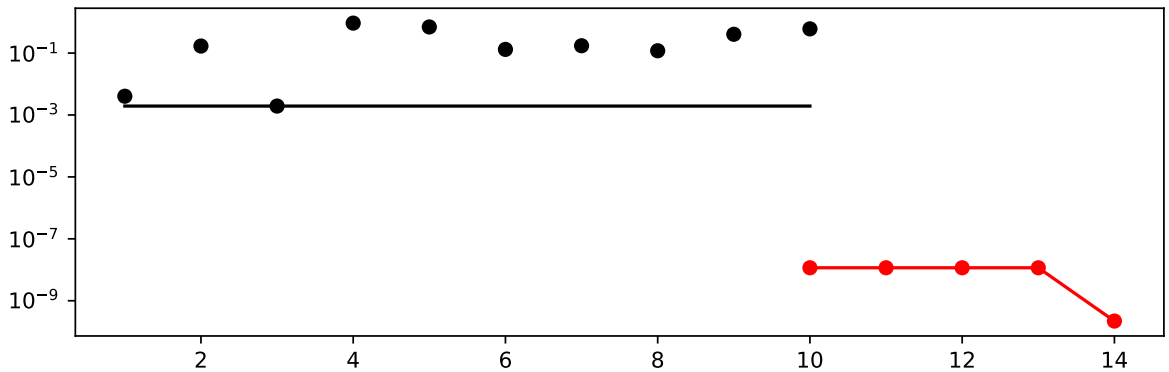



## 7.2 Same, but with EI as infill_criterion

```
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 0x14f7ecca0>
```

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




```
spot_1_ei.print_results()
```

```
min y: 2.207887258868953e-10
x0: 1.4858961130809088e-05

[['x0', 1.4858961130809088e-05]]
```

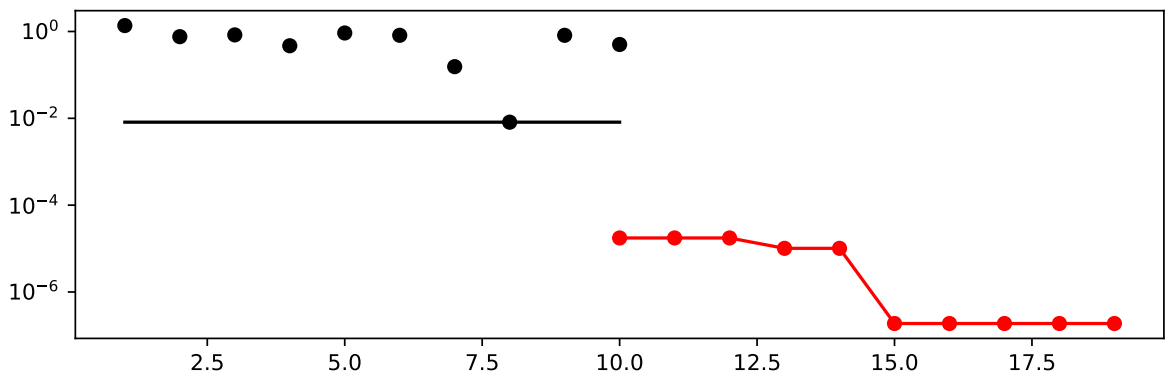
## 7.3 Non-isotropic Kriging

```
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 0x16a01ab90>
```

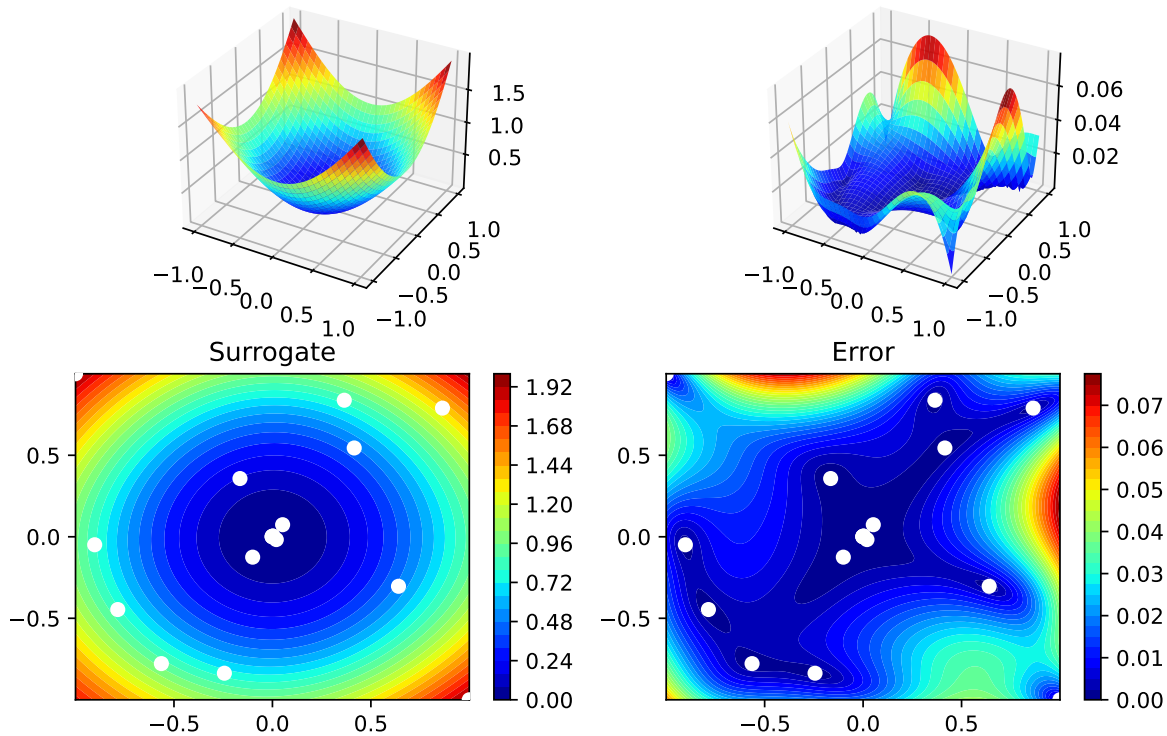
```
spot_2_ei_noniso.plot_progress(log_y=True)
```




```
spot_2_ei_noniso.print_results()
```

```
min y: 1.8779971830281702e-07
x0: -0.0002783721390529846
x1: 0.0003321274913371111

[['x0', -0.0002783721390529846], ['x1', 0.0003321274913371111]]
```

```
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:

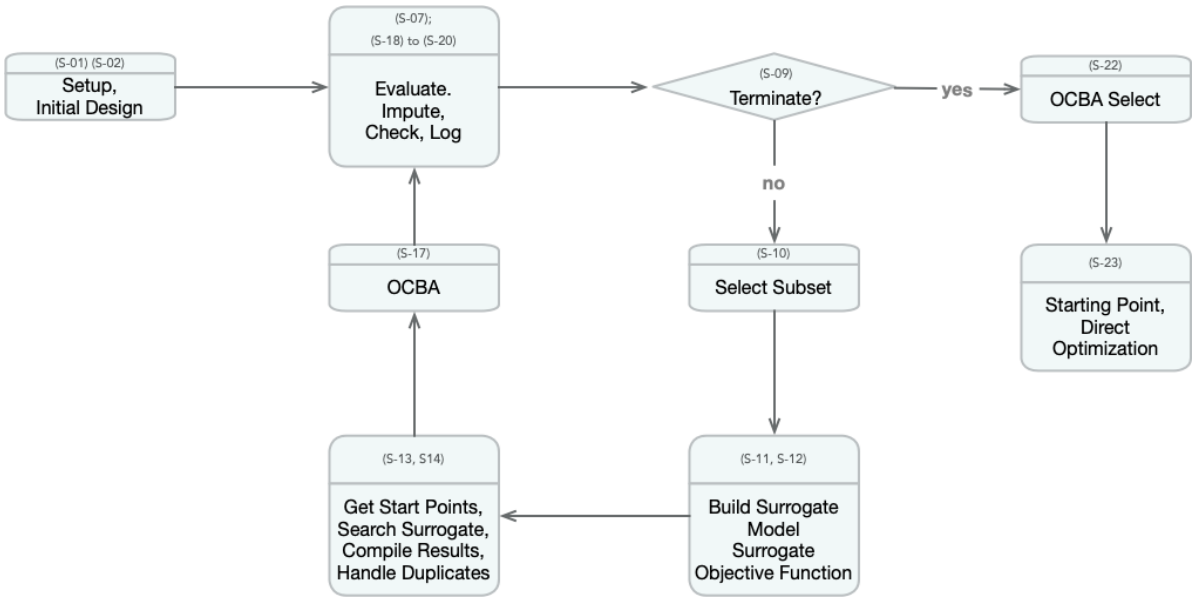



Figure 7.1: Visual representation of the model based search with SPOT. Taken from: Bartz-Beielstein, T., and Zaefferer, M. Hyperparameter tuning approaches. In Hyperparameter Tuning for Machine and Deep Learning with R - A Practical Guide, E. Bartz, T. Bartz-Beielstein, M. Zaefferer, and O. Mersmann,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

### 7.4.2 spot: The Initial Model

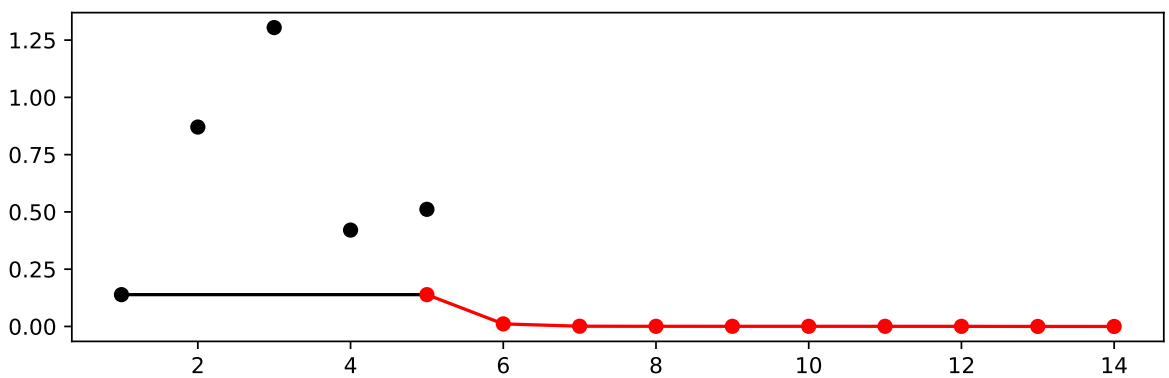
#### 7.4.2.1 Example: Modifying the initial design size

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 0x16a622140>
```

```
spot_ei.plot_progress()
```




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

```
(3.696886711914087e-10, 1.7928640814182596e-05)
```

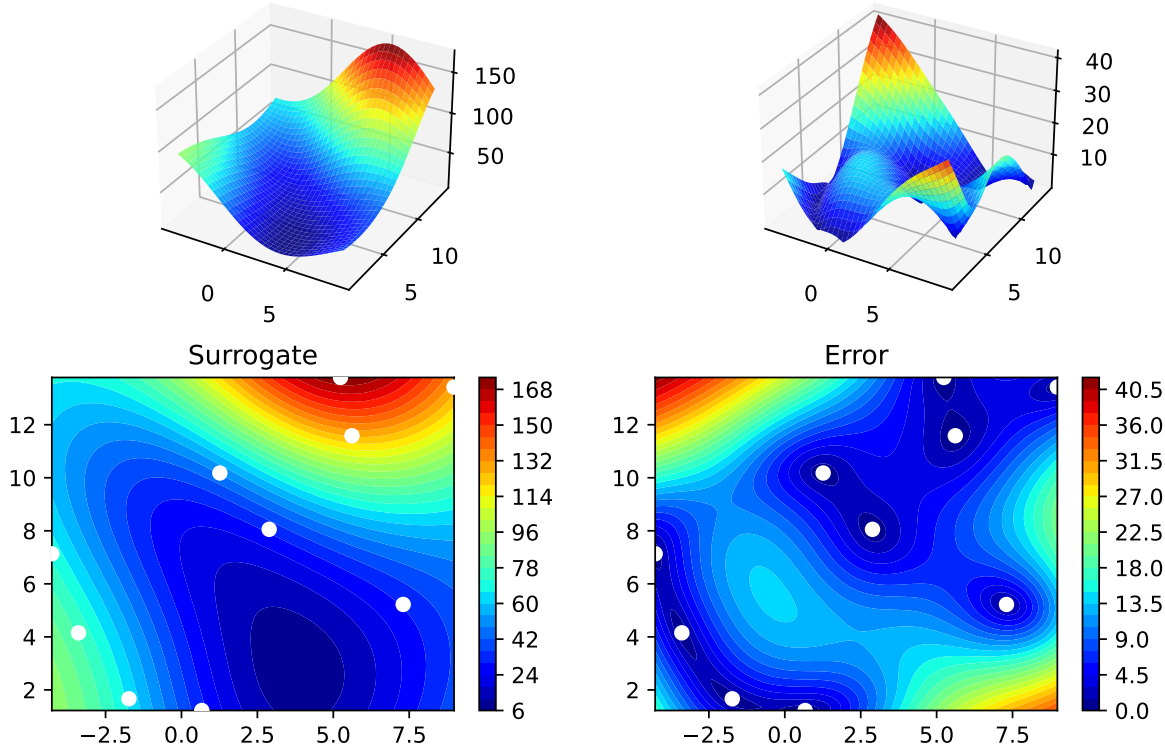
### 7.4.3 Init: Build Initial Design

```
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)
```

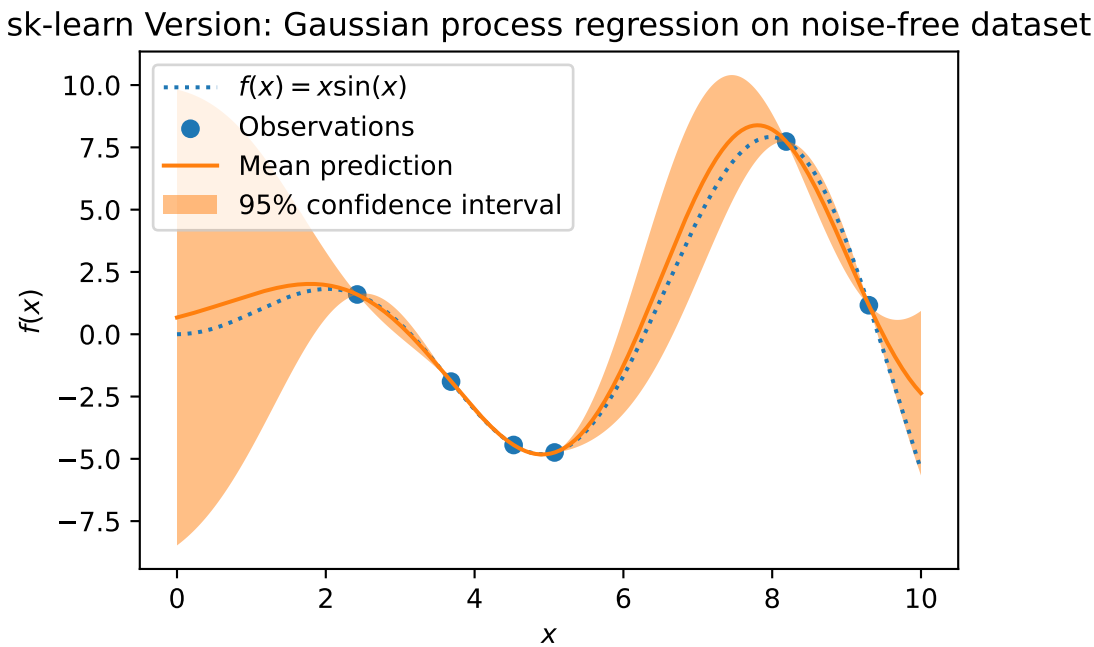
```
[1.5]
```

- 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

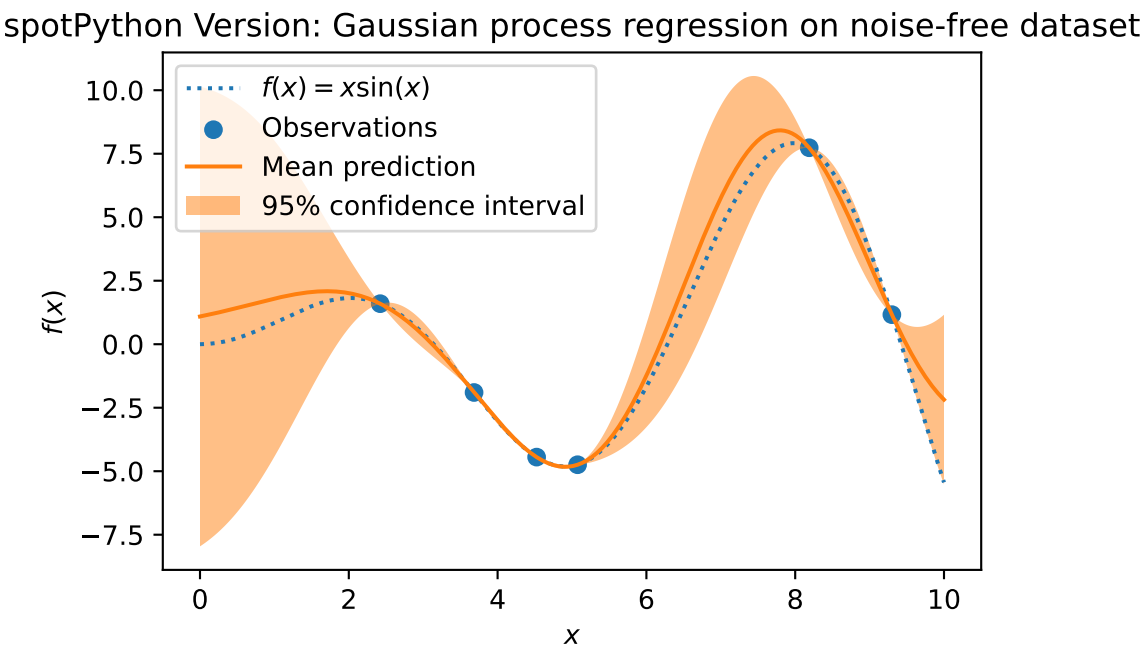
This example was taken from scikit-learn. After fitting our model, we see that the hyperparameters of the kernel have been optimized. Now, we will use our kernel to compute the mean prediction of the full dataset and plot the 95% confidence interval.

```python
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")
```




```
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")
```




## 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:

```python
# 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
```

```python
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:

```python
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.

```python
S = S_GP
```

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

```
True
```

```python
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)
```

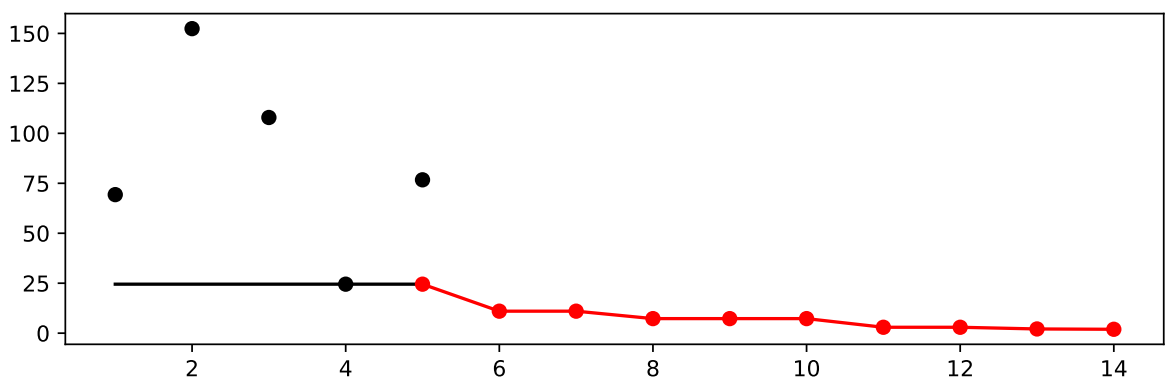
```
spot_GP.run()
```

```
<spotPython.spot.spot.Spot at 0x16a7cb4f0>
```

```
spot_GP.y
```

```
array([ 69.32459936, 152.38491454, 107.92560483,  24.51465459,
        76.73500031,  86.30426868,  11.00311217,  16.11744031,
         7.2812424 ,  21.82314438,  10.96088904,   2.95186993,
         3.02908308,   2.10496308,   1.94329383])
```

```
spot_GP.plot_progress()
```




```
spot_GP.print_results()
```

```
min y: 1.943293826587226
x0: 9.999957143880025
x1: 2.983496927641645

[['x0', 9.999957143880025], ['x1', 2.983496927641645]]
```

## 7.7 Additional Examples

```python
# 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
```

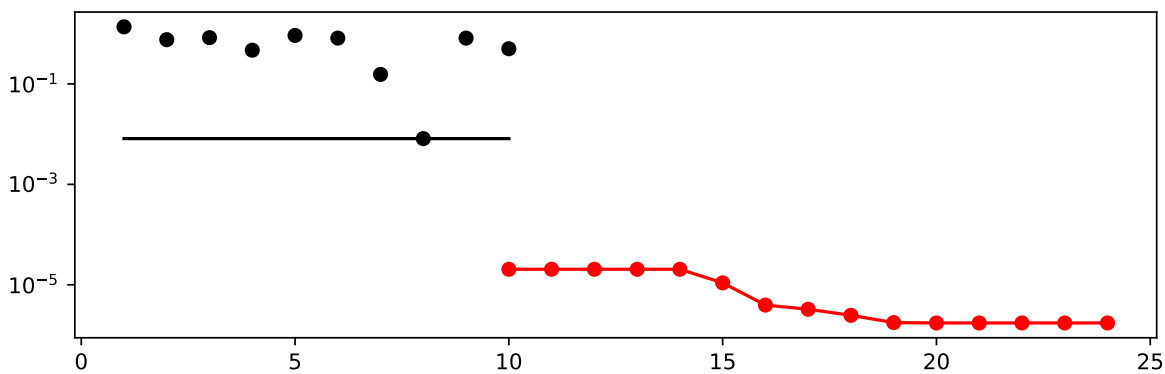
```python
kernel = 1 * RBF(length_scale=1.0, length_scale_bounds=(1e-2, 1e2))
S_GP = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=9)
```

```python
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,
```

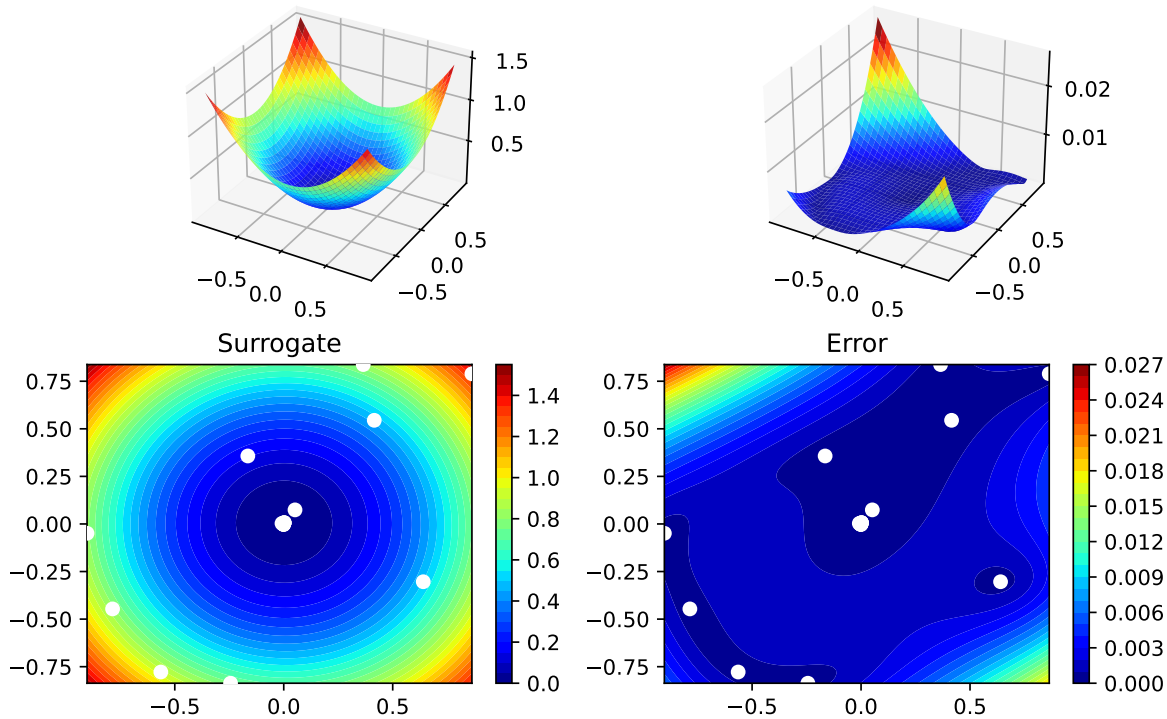
```
                        design_control=design_control,
                        surrogate_control=surrogate_control)

spot_S_K.run()
```

```
<spotPython.spot.spot.Spot at 0x10ff33f70>
```

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




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




```
spot_S_K.print_results()
```

```
min y: 1.7395335905335862e-06
x0: -0.0013044072412622557
x1: 0.0001950777780173277

[['x0', -0.0013044072412622557], ['x1', 0.0001950777780173277]]
```

### 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]
```

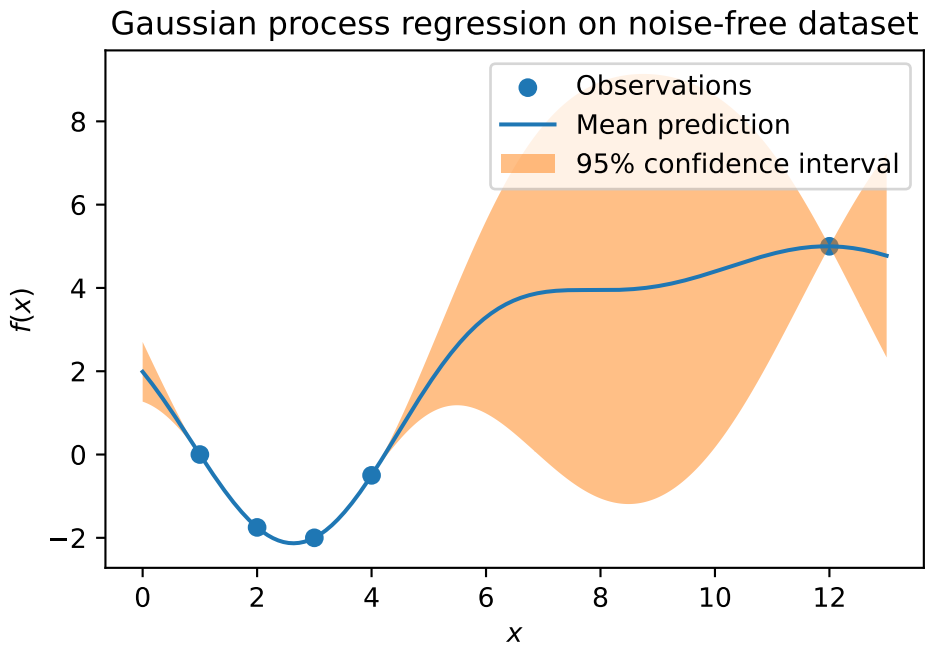
```
 [-0.16484832  0.35724741]
 [ 0.05170659  0.07401196]
 [-0.78548145 -0.44638164]
 [ 0.64017497 -0.30363301]]
[1.36857656 0.75992983 0.83463487 0.46918172 0.92329124 0.8170764
 0.15480068 0.00815134 0.81623768 0.502017  ]
[[0.00160553 0.00428429]
 [0.00160553 0.00428429]]
```

## 7.9 EI: The Famous Schonlau Example

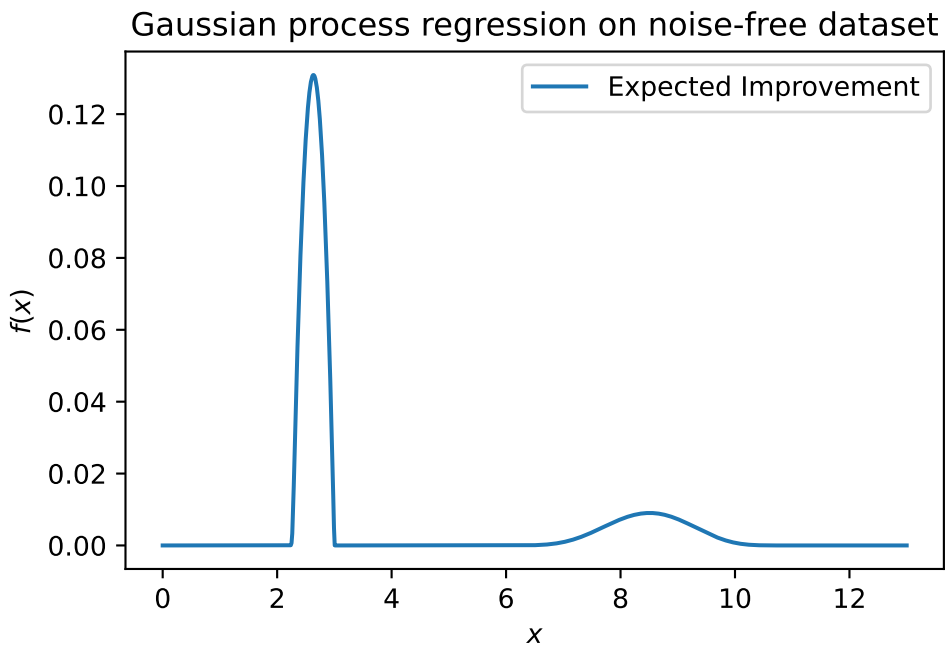
```
X_train0 = np.array([1, 2, 3, 4, 12]).reshape(-1,1)
X_train = np.linspace(start=0, stop=10, num=5).reshape(-1, 1)
```

```
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="nor
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")
```



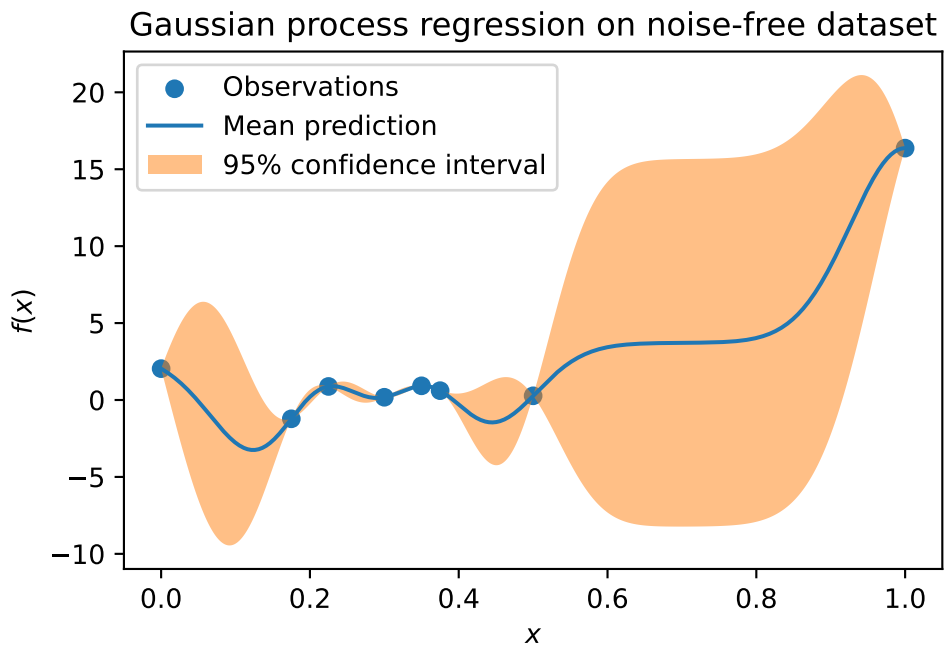

```
S.log
```

```
{'negLnLike': array([1.20788205]),
 'theta': array([1.09276]),
 'p': array([2.]),
 'Lambda': array([None], dtype=object)}
```

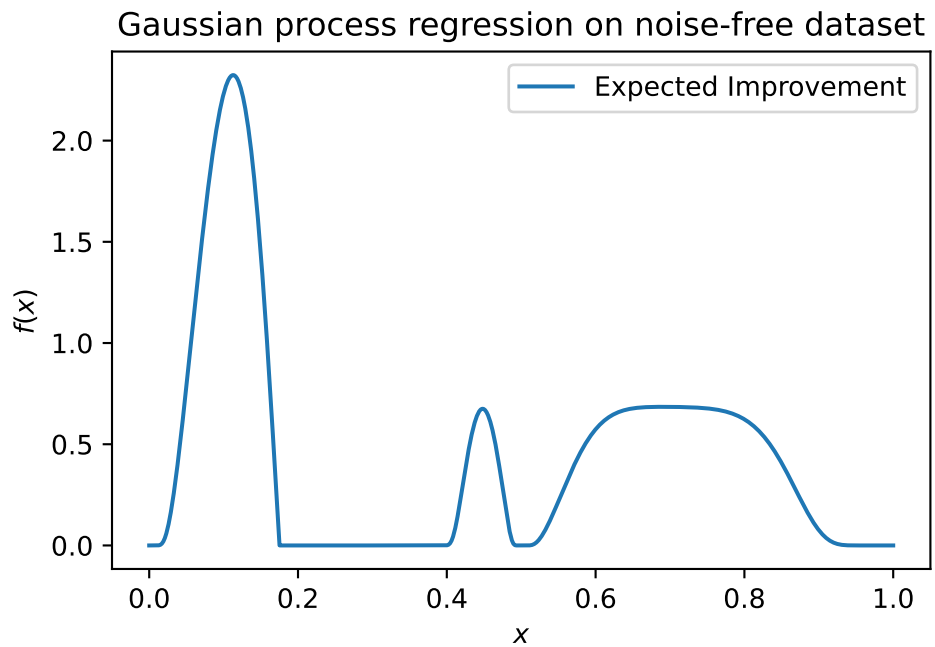
## 7.10 EI: The Forrester Example

```
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="nor
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")
```

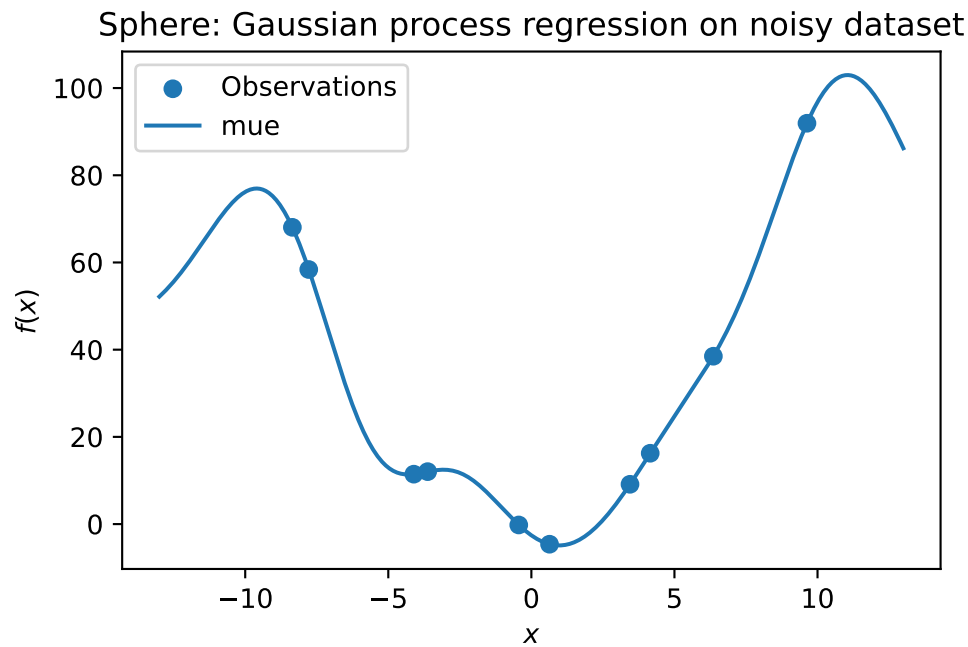



## 7.11 Noise

```
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]
```

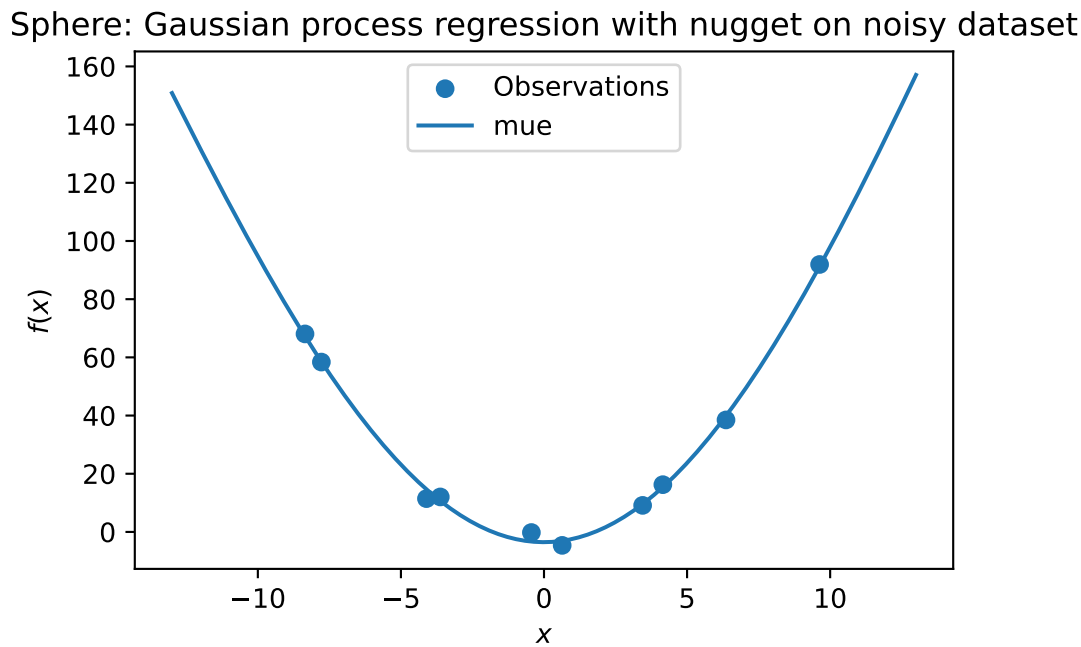



```
S.log
```

```
{'negLnLike': array([24.69806131]),
 'theta': array([1.31023943]),
 '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")
```




```
S.log
```

```
{'negLnLike': array([22.14095646]),
 'theta': array([-0.32527397]),
 'p': array([2.]),
 'Lambda': array([9.08815007e-05])}
```

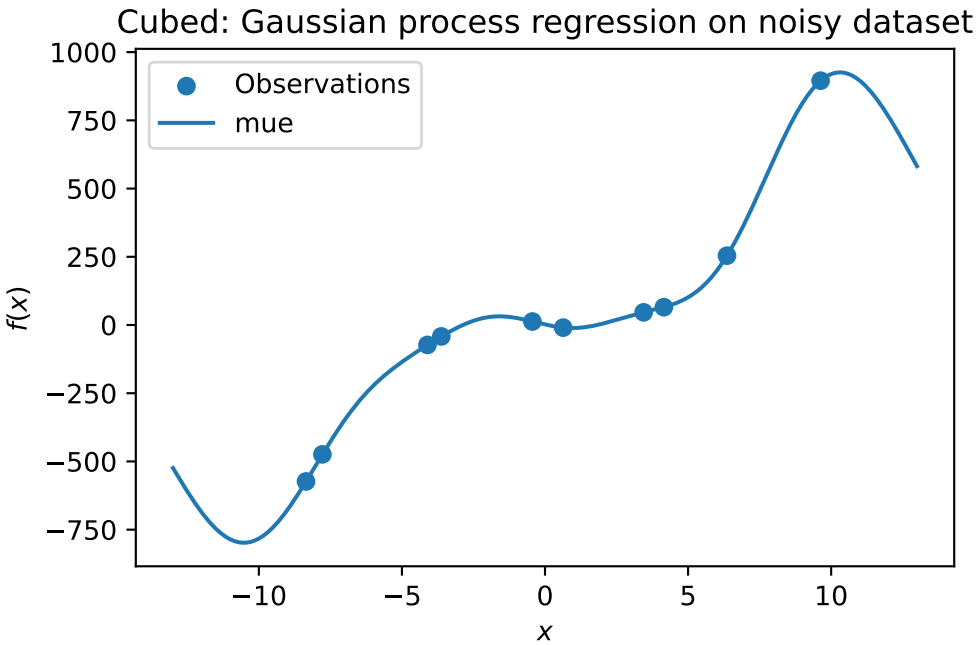
## 7.12 Cubic Function

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

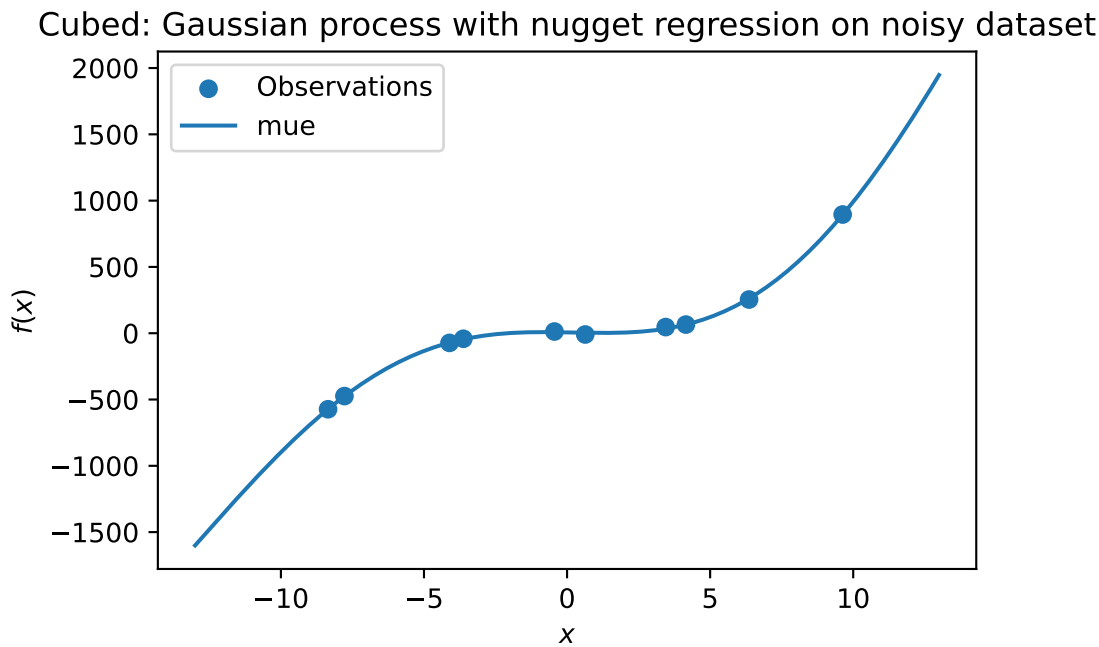
```python
from spotPython.build.kriging import Kriging
import matplotlib.pyplot as plt

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]
```



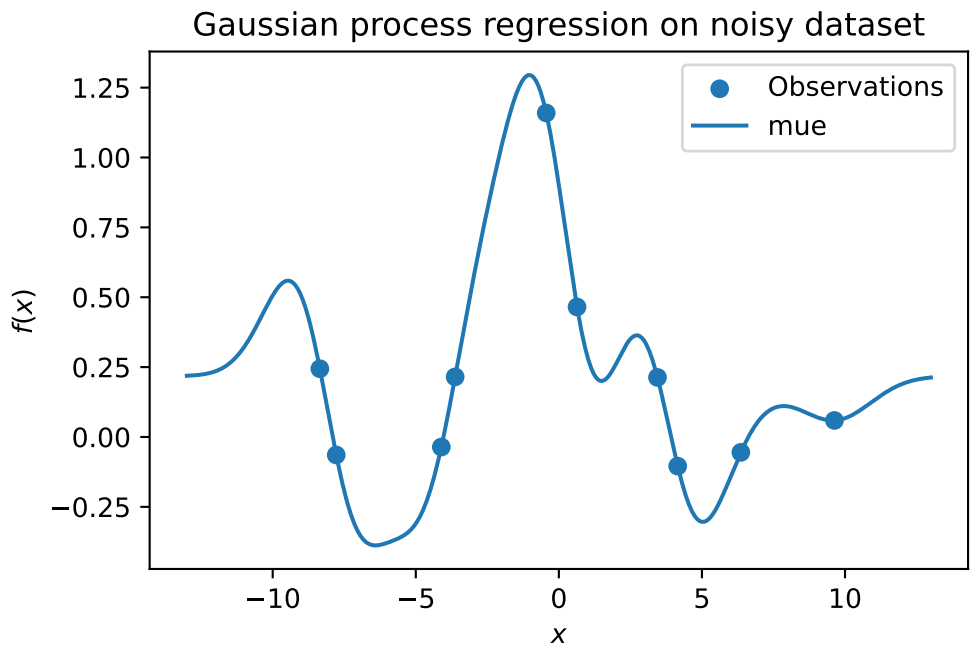

```
S = Kriging(name='kriging',  seed=123, log_level=0, n_theta=1, noise=True)
S.fit(X_train, y_train)

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")
```



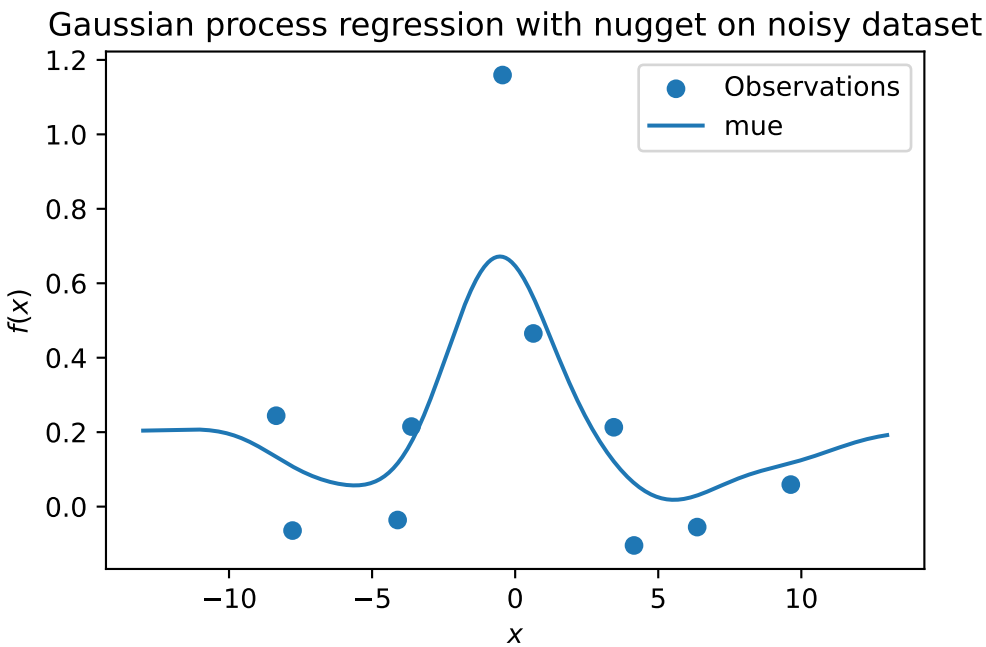

```
import numpy as np
import spotPython
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from spotPython.design.spacefilling import spacefilling
from spotPython.build.kriging import Kriging
import matplotlib.pyplot as plt

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
```

```python
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
```

```
87.44231855765292
```

```
# 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_p040025_2023-07-06_00-47-01
```

### 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:

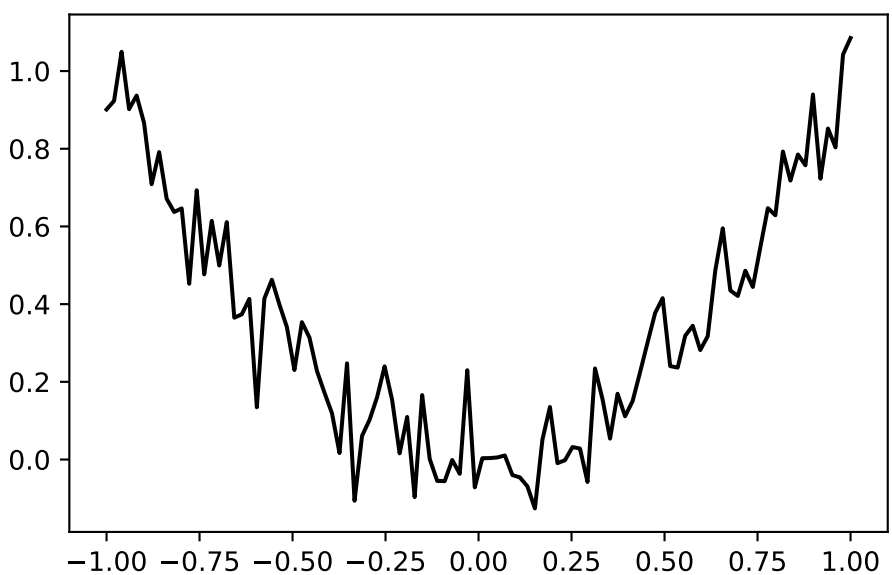
$$f(x) = x^2 + \epsilon$$

- 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()
```

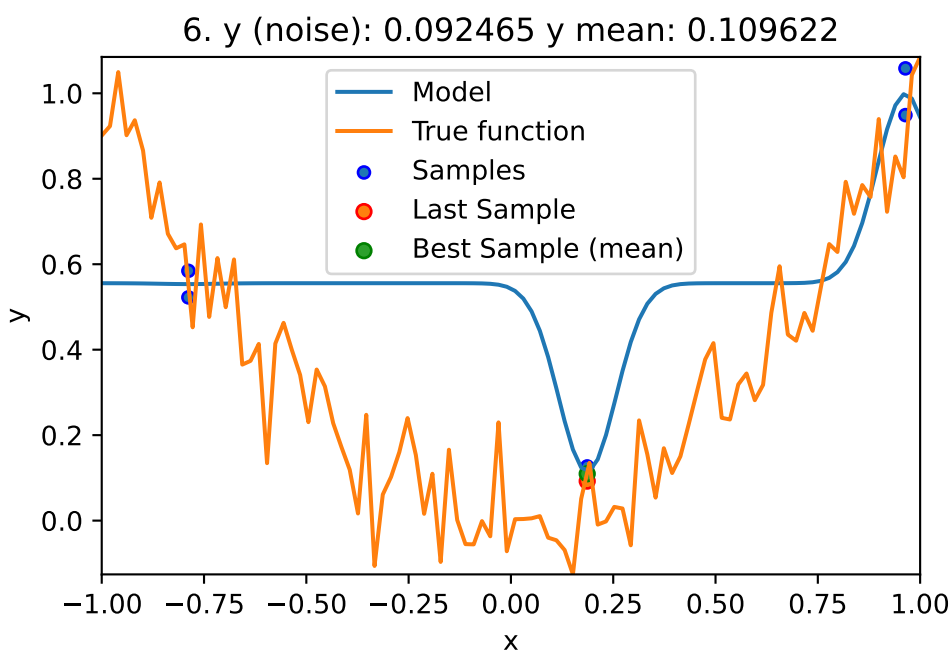


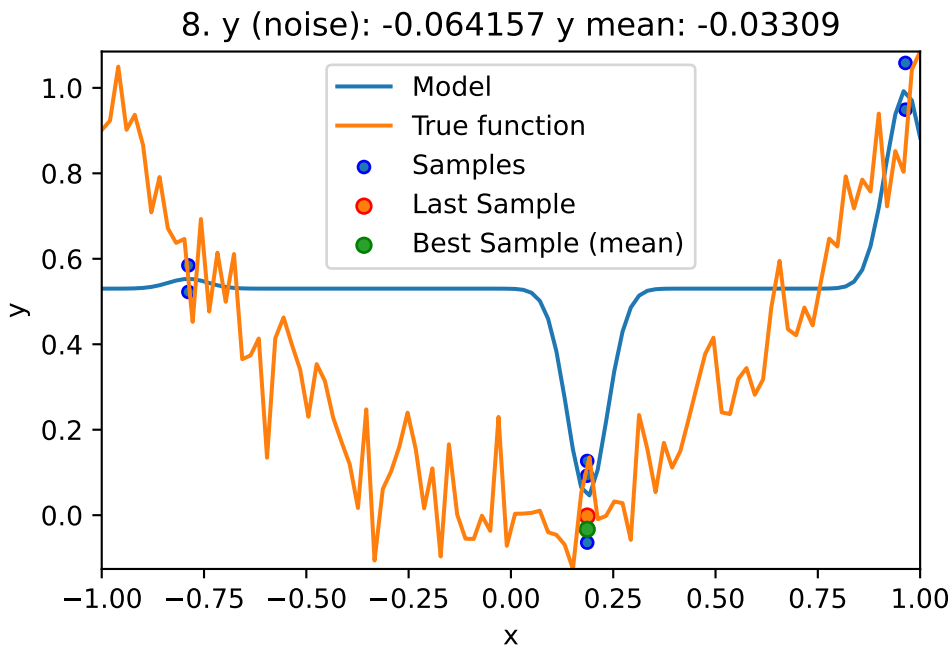

`Spot` is adopted as follows to cope with noisy functions:

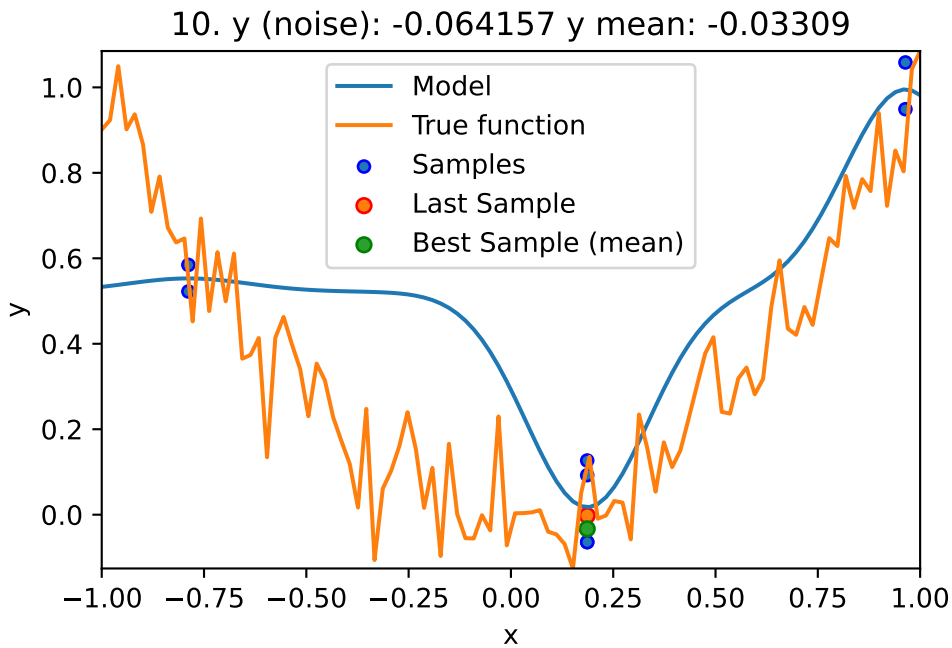
1. `fun_repeats` is set to a value larger than 1 (here: 2)
2. `noise` is set to `true`. Therefore, a nugget (`Lambda`) term is added to the correlation matrix
3. `init size` (of the `design_control` dictionary) is set to a value larger than 1 (here: 2)

```
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 0x1433e8040>
```

## 8.2 Print the Results

```
spot_1_noisy.print_results()
```

```
min y: -0.06415721594238855
x0: 0.18642671238960512
min mean y: -0.03309048099839016
x0: 0.18642671238960512

[['x0', 0.18642671238960512], ['x0', 0.18642671238960512]]
```

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

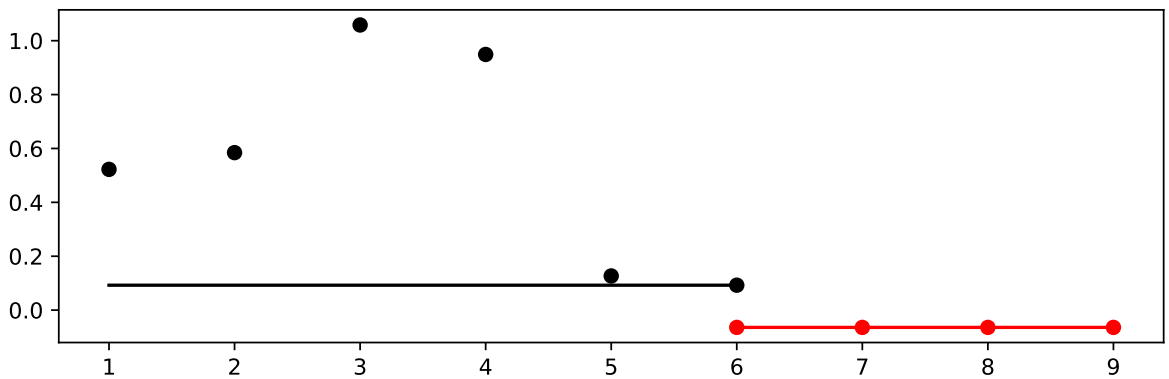



Figure 8.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

## 8.3 Noise and Surrogates: The Nugget Effect

### 8.3.1 The Noisy Sphere

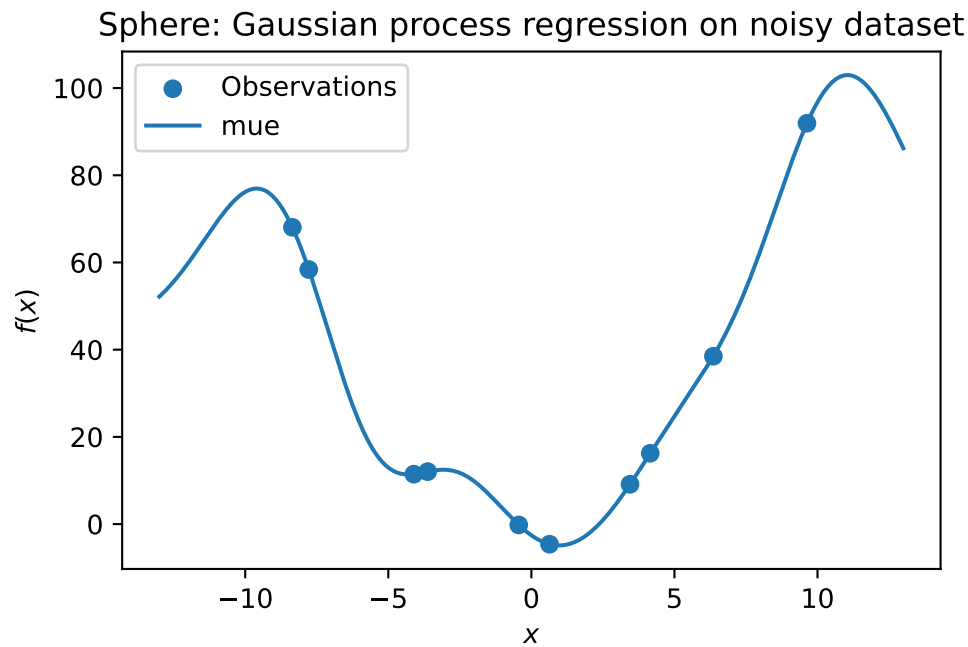
#### 8.3.1.1 The Data

- 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
```

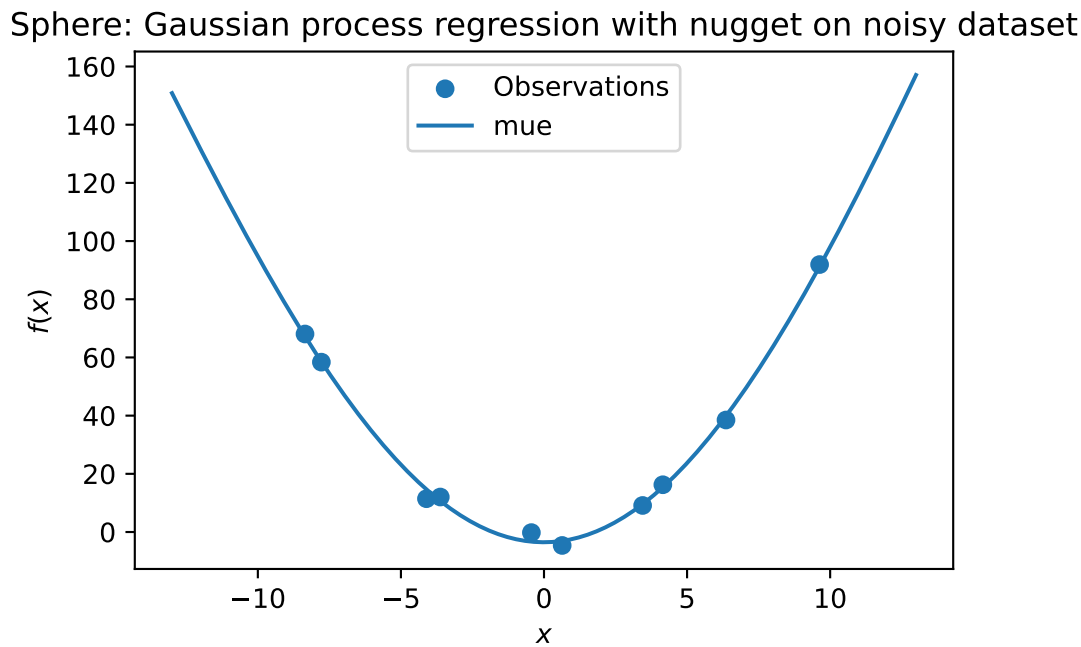
- A surrogate without nugget is fitted to these data:

```
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")
```




- 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")
```




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

```
S.Lambda
```

```
S_nug.Lambda
```

```
9.088150066416743e-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:

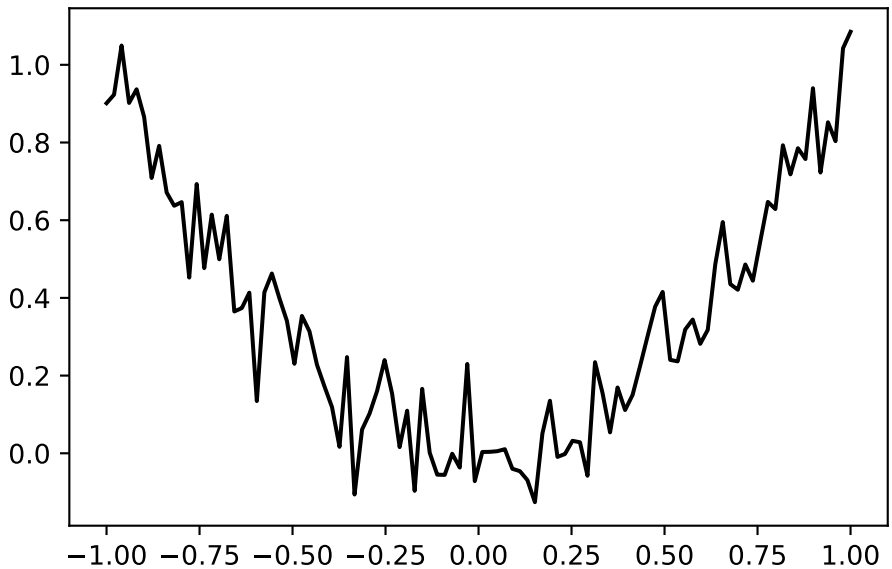
$$f(x) = x^2 + \epsilon$$

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()
```



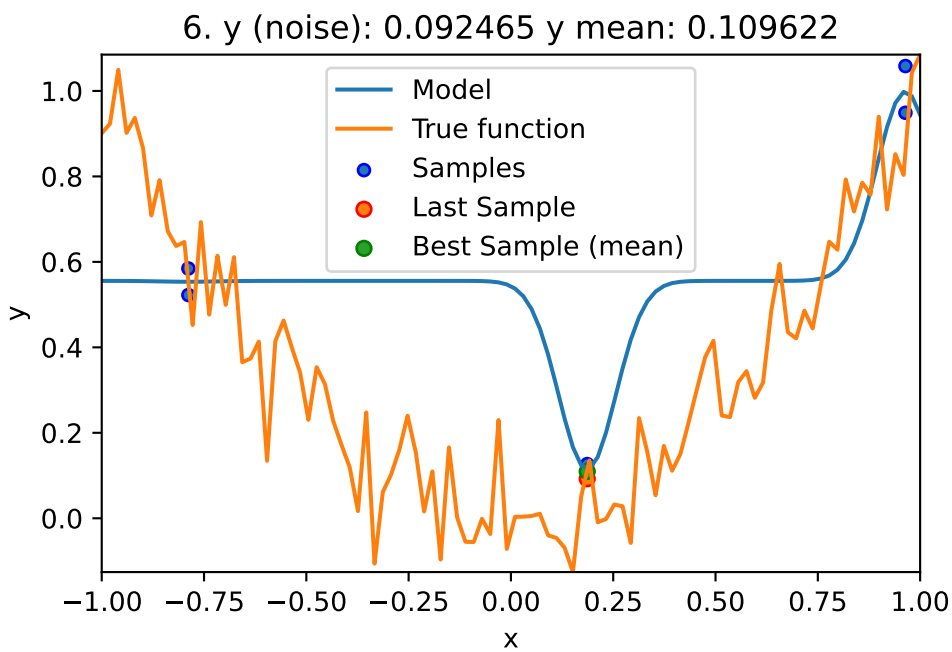

`Spot` is adopted as follows to cope with noisy functions:

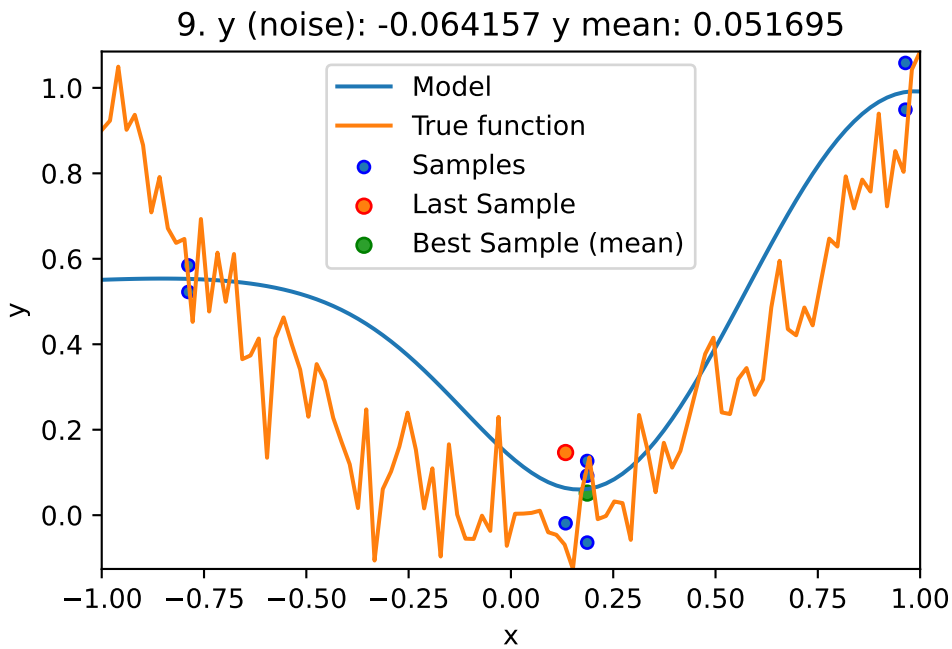
1. `fun_repeats` is set to a value larger than 1 (here: 2)
2. `noise` is set to `true`. Therefore, a nugget (`Lambda`) term is added to the correlation matrix
3. `init size` (of the `design_control` dictionary) is set to a value larger than 1 (here: 2)

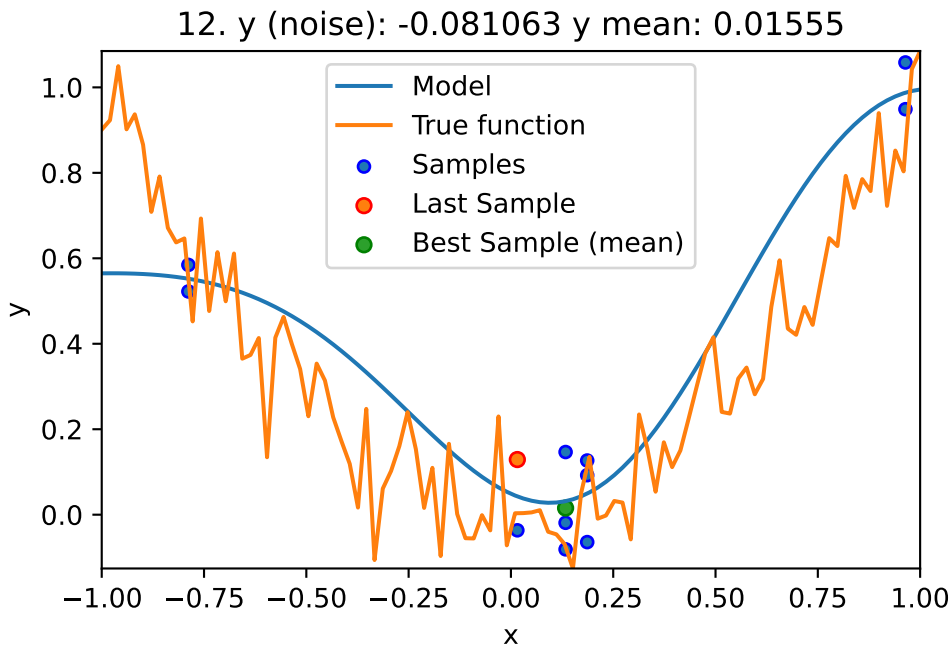
```
spot_1_noisy = spot.Spot(fun=fun,
                    lower = np.array([-1]),
                    upper = np.array([1]),
                    fun_evals = 50,
                    fun_repeats = 2,
                    infill_criterion="ei",
                    noise = True,
                    tolerance_x=0.0,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

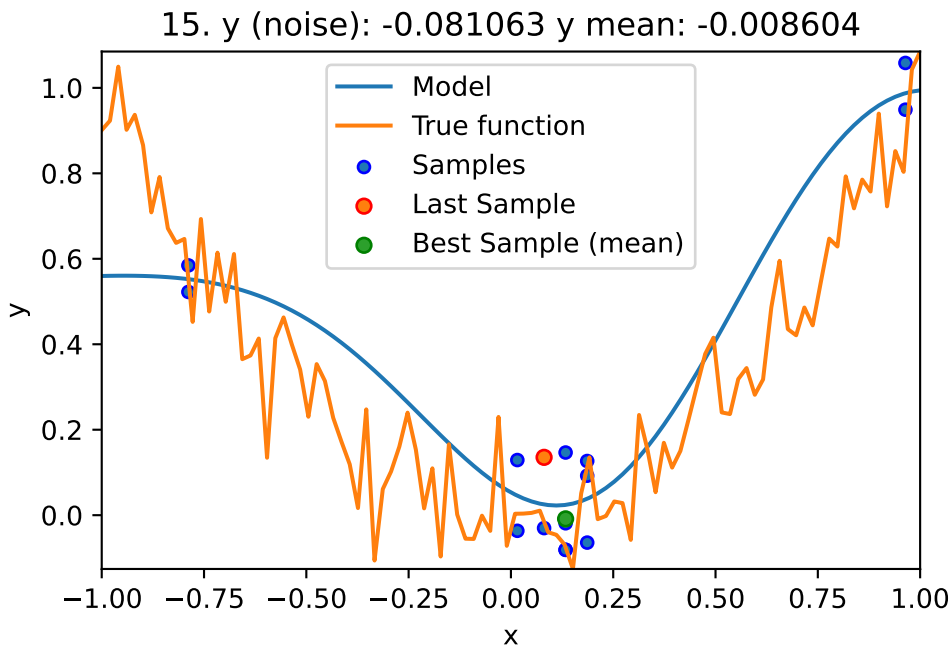
```
                    show_models=True,
                    fun_control = fun_control,
                    design_control={"init_size": 3,
                                    "repeats": 2},
                    surrogate_control={"noise": True})
```

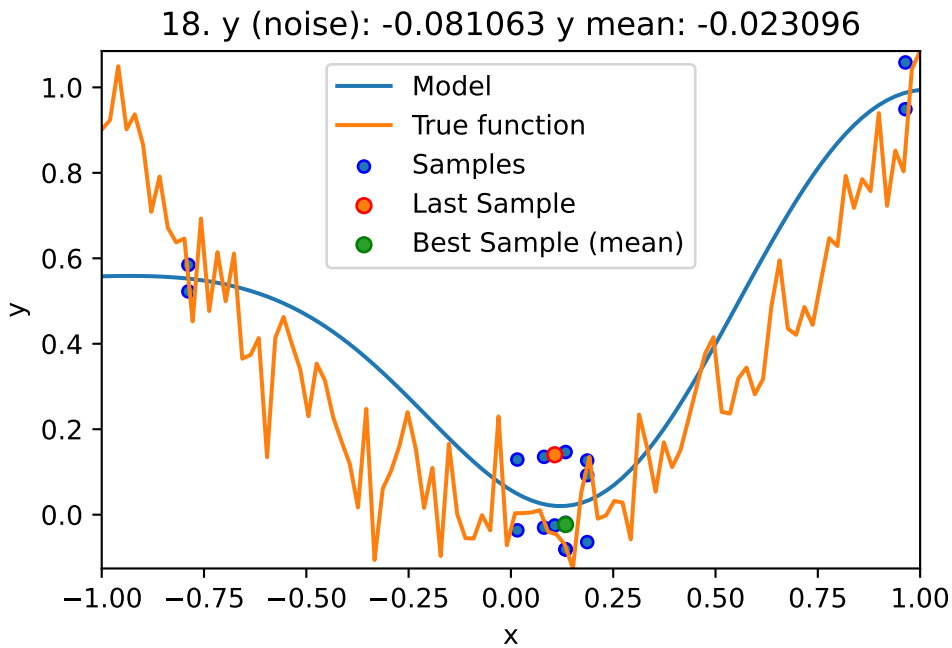
```
spot_1_noisy.run()
```

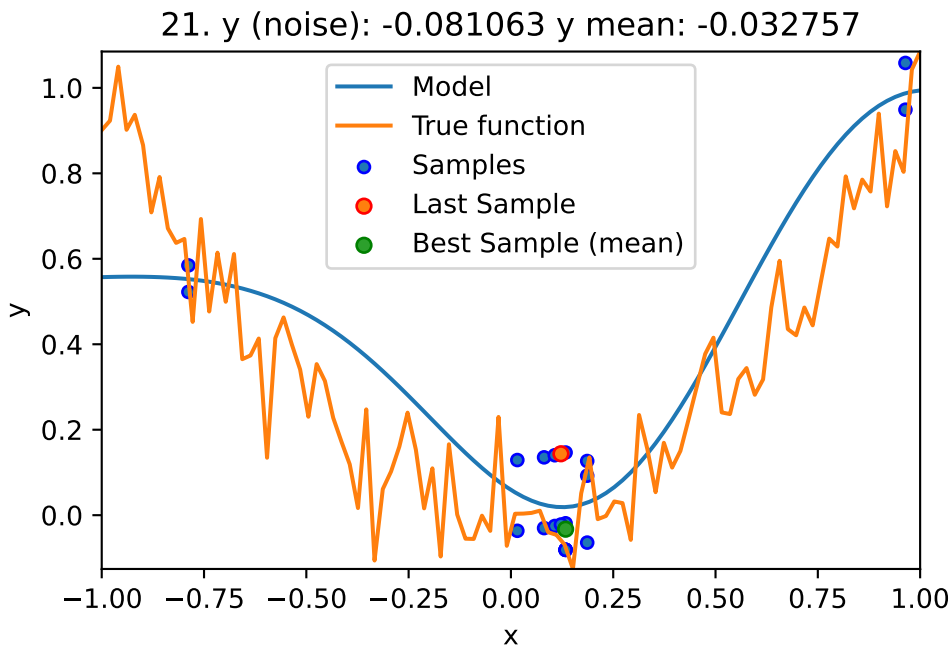


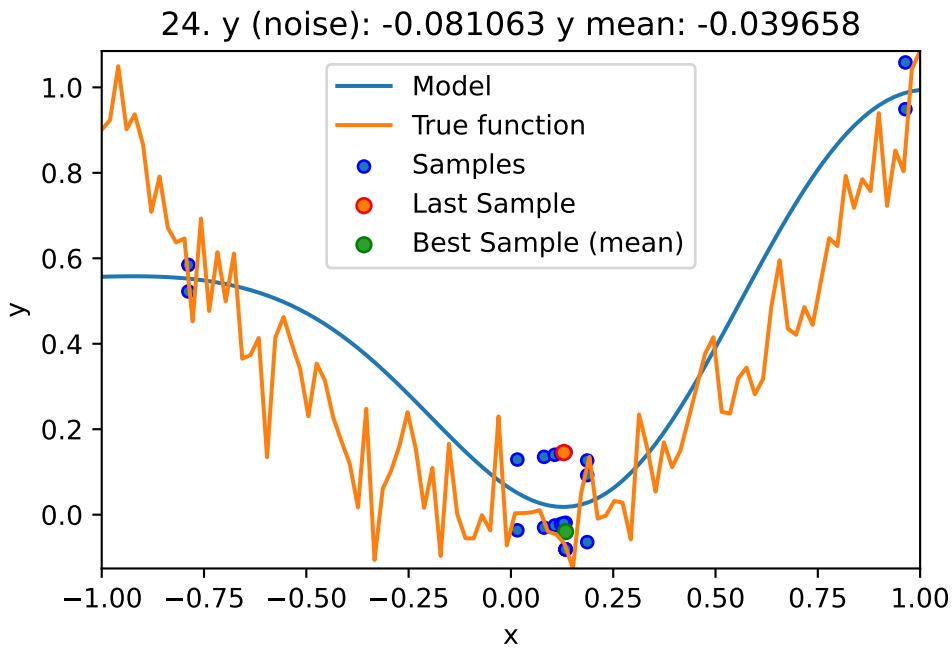

9. y (noise): -0.064157 y mean: 0.051695
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

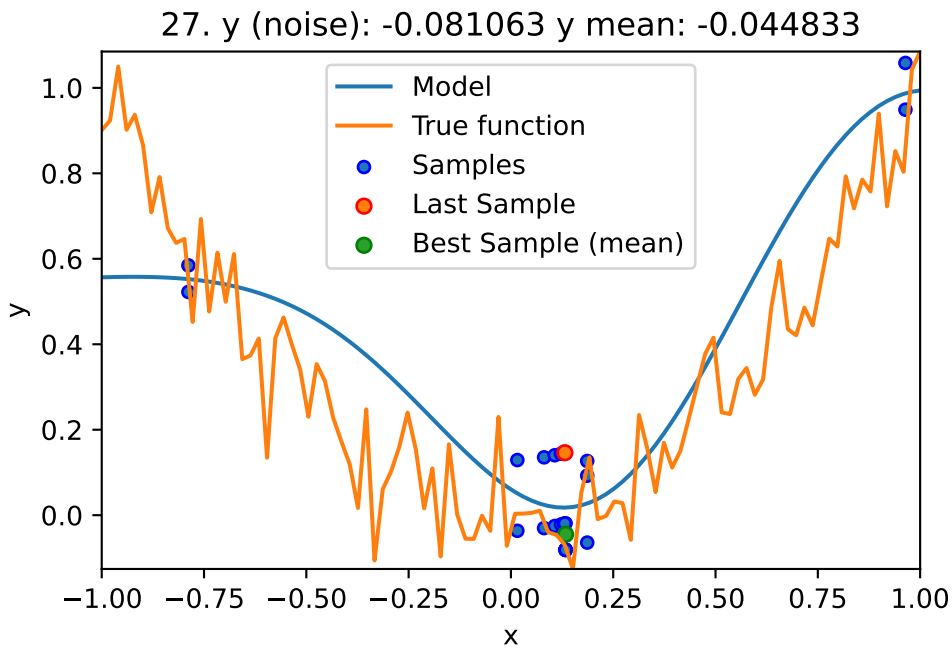

12. y (noise): -0.081063 y mean: 0.01555
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

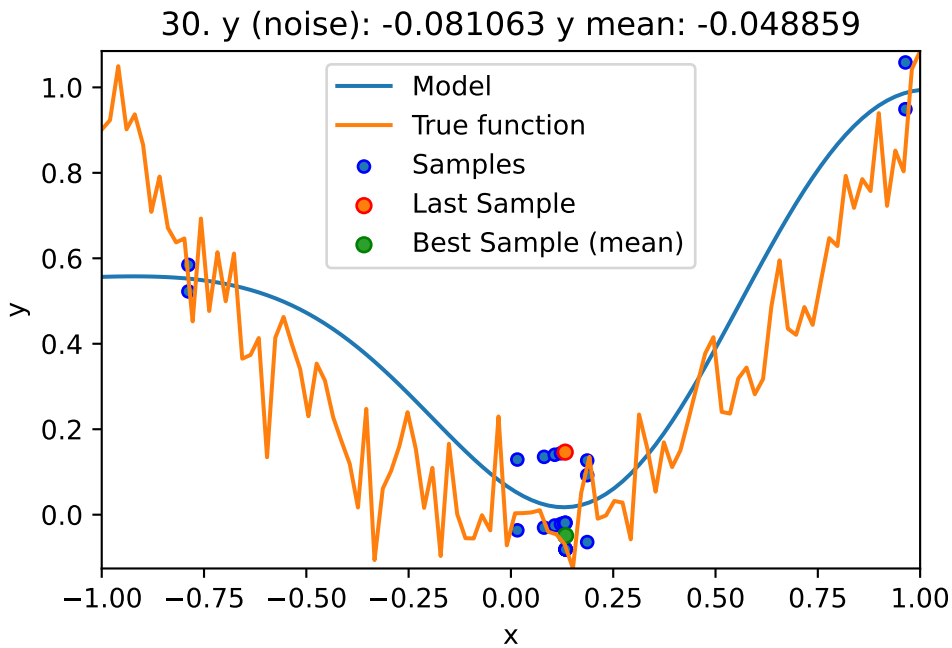

15. y (noise): -0.081063 y mean: -0.008604
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

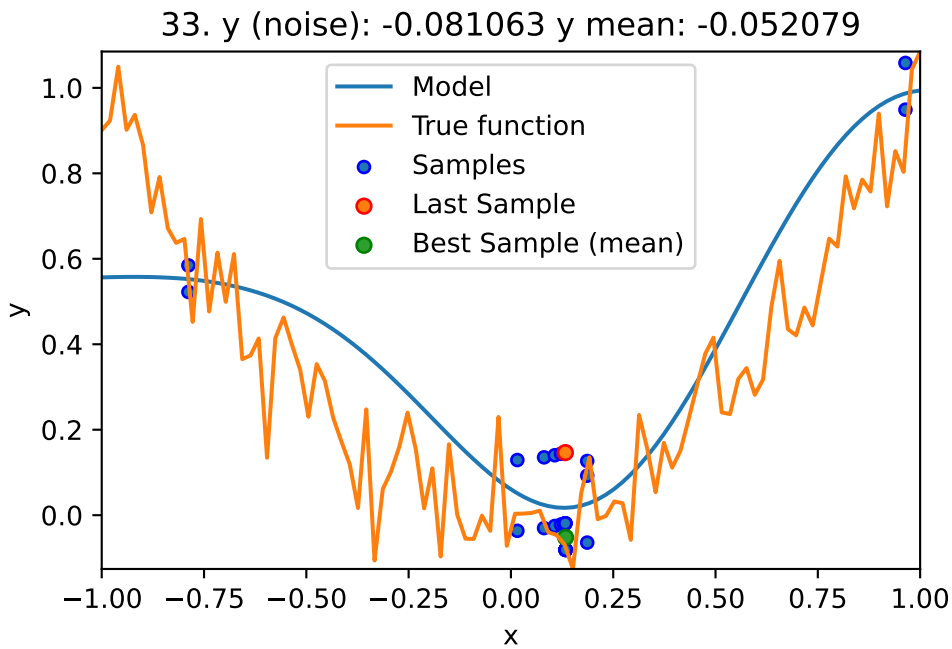

18. y (noise): -0.081063 y mean: -0.023096
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

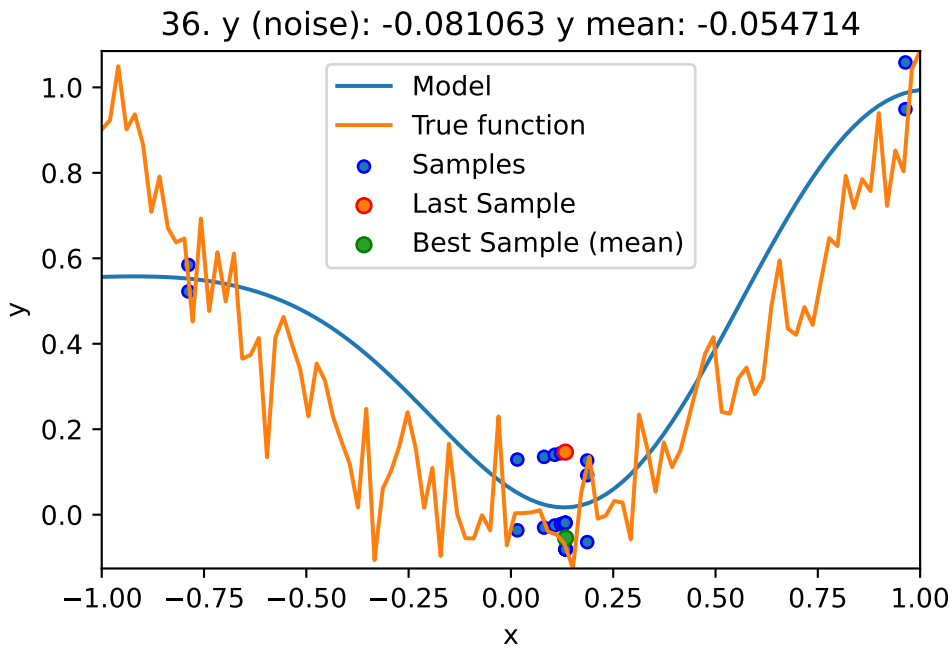

21. y (noise): -0.081063 y mean: -0.032757
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

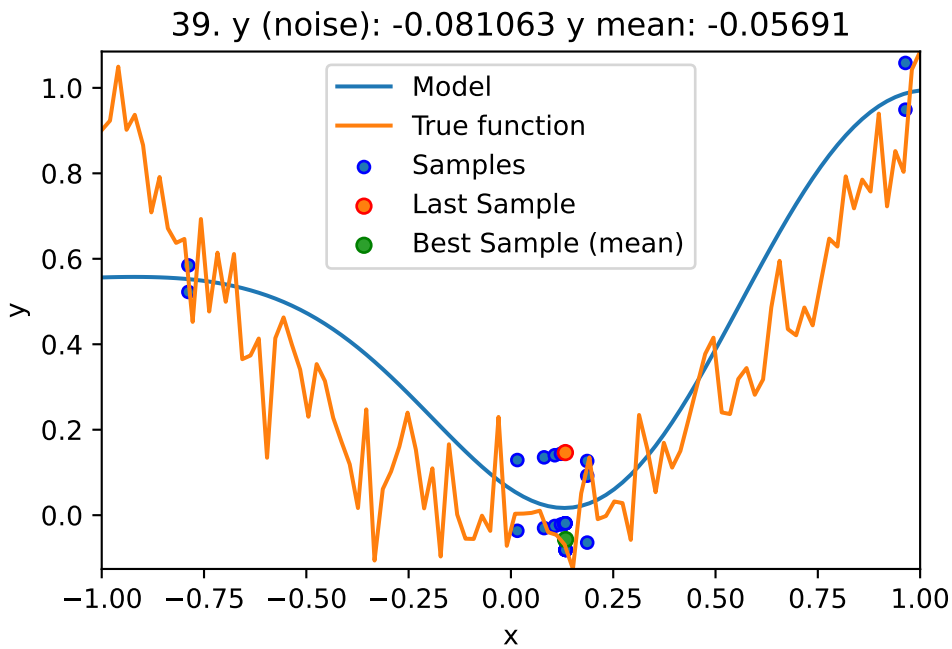

24. y (noise): -0.081063 y mean: -0.039658
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

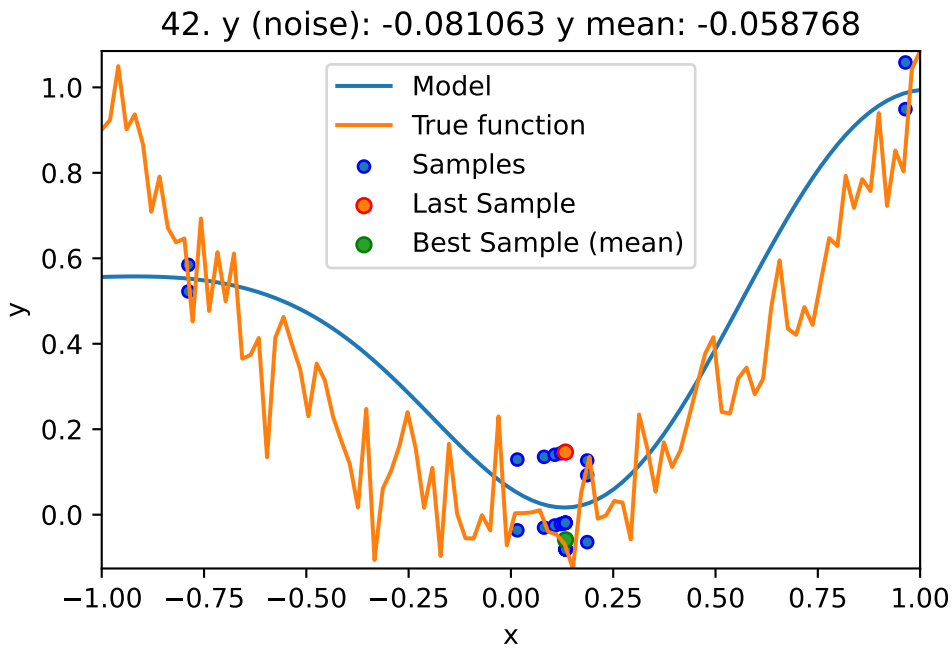

27. y (noise): -0.081063 y mean: -0.044833
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

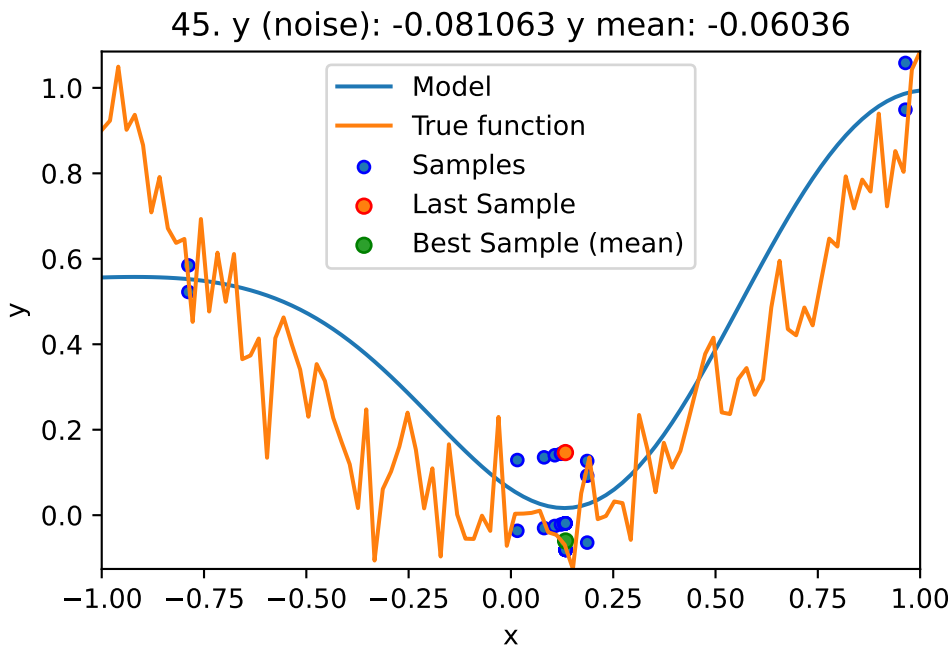

30. y (noise): -0.081063 y mean: -0.048859
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

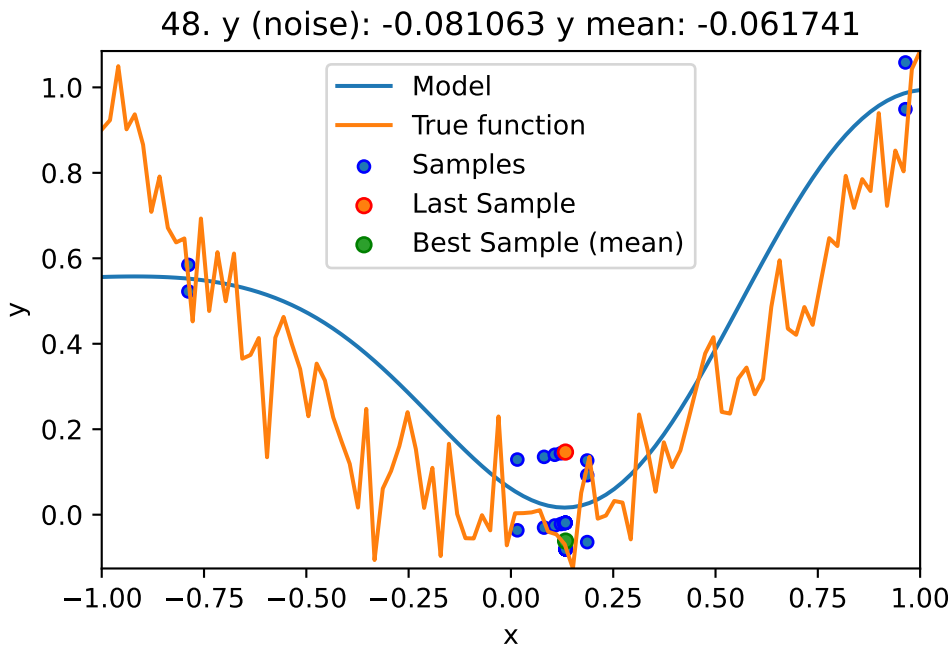

33. y (noise): -0.081063 y mean: -0.052079
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

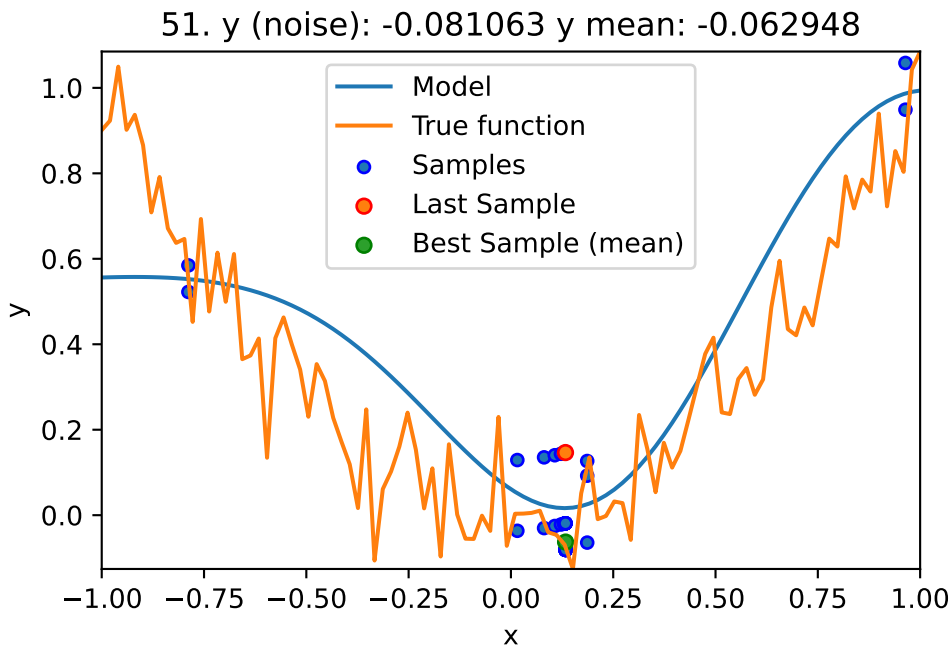

36. y (noise): -0.081063 y mean: -0.054714
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


39. y (noise): -0.081063 y mean: -0.05691
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


42. y (noise): -0.081063 y mean: -0.058768
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


45. y (noise): -0.081063 y mean: -0.06036
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


48. y (noise): -0.081063 y mean: -0.061741
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00



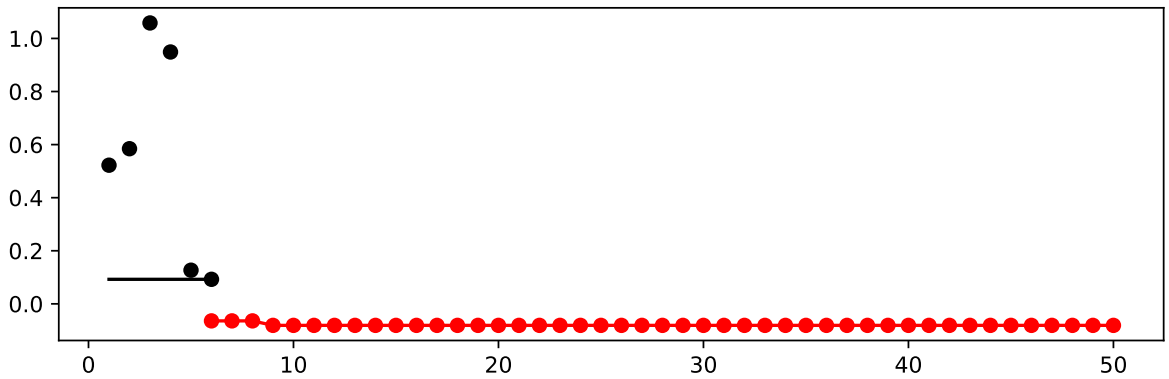

```
<spotPython.spot.spot.Spot at 0x12f324430>
```

## 9.2 Print the Results

```
spot_1_noisy.print_results()
```

```
min y: -0.08106318979661208
x0: 0.1335999447536301
min mean y: -0.06294830660588041
x0: 0.1335999447536301

[['x0', 0.1335999447536301], ['x0', 0.1335999447536301]]
```

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




## 9.3 Noise and Surrogates: The Nugget Effect

### 9.3.1 The Noisy Sphere

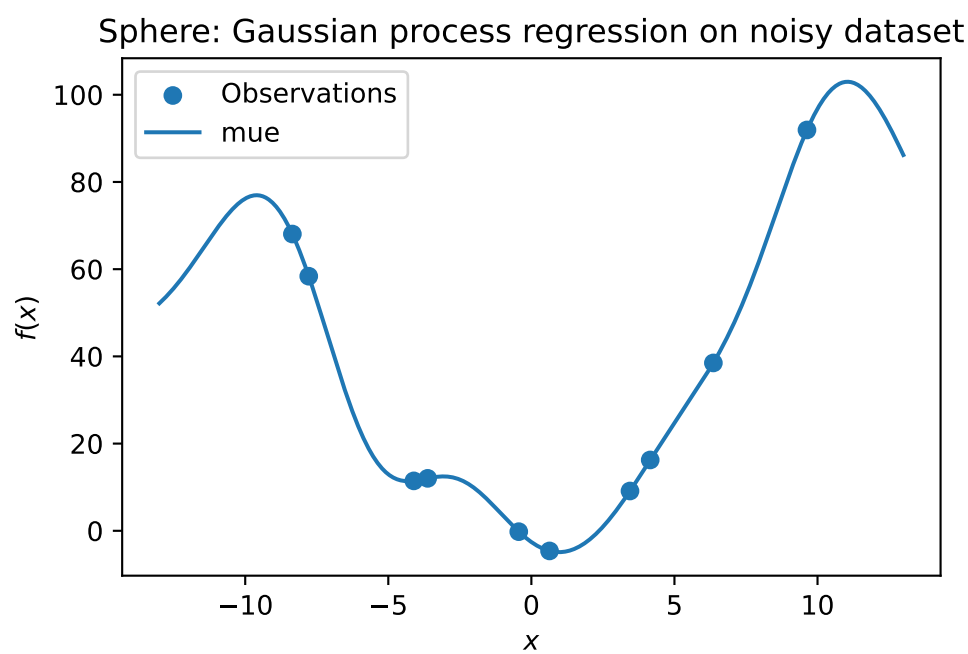
#### 9.3.1.1 The Data

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
```

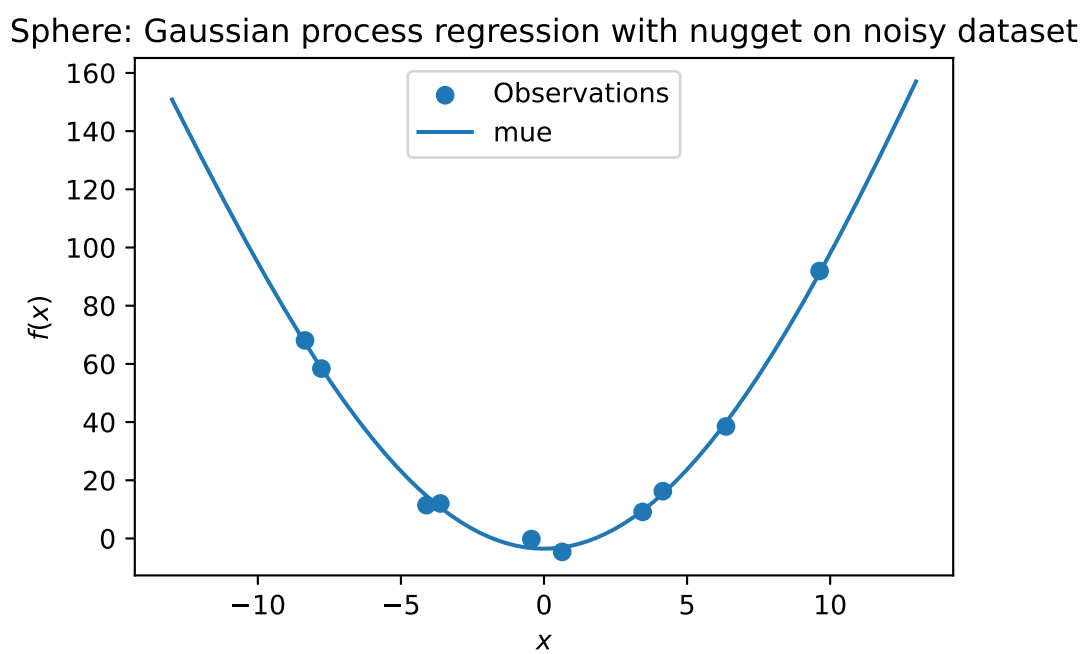
A surrogate without nugget is fitted to these data:

```
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")
```




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

```
S.Lambda
```

```
S_nug.Lambda
```

```
9.088150066416743e-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]
```

```
18
```

# 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                  0.3.4
spotRiver                   0.0.93

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_p040025_1min_5init_2023-07-06_00-48-38
```

## 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)

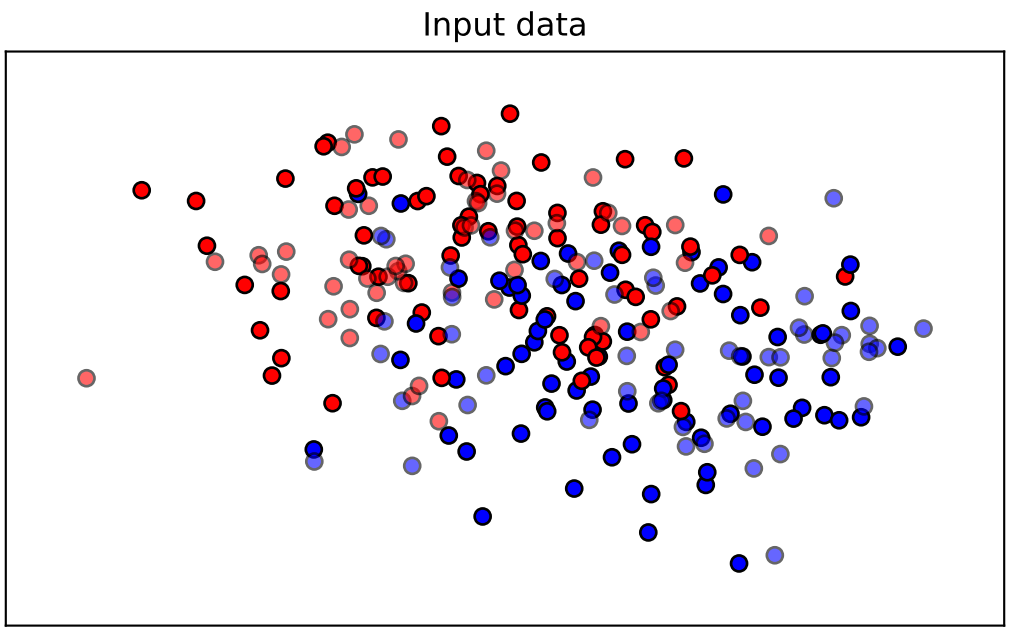
Randomly generate classification data.

```
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()
```




```
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 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:

```
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 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(),
#     )
```

## 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
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 12.6.

### 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
modify_hyper_parameter_bounds(fun_control, "tol", bounds=[1e-3, 1e-2])
# modify_hyper_parameter_bounds(fun_control, "min_samples_split", bounds=[3, 20])
#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
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 12.6.1.

## 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)
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: `initi_size`, 20 points) is not considered.

```
from spotPython.hyperparameters.values import get_default_hyperparameters_as_array
X_start = get_default_hyperparameters_as_array(fun_control)
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 [----------] 2.87%

spotPython tuning: 5.691103166702708 [----------] 4.72%

spotPython tuning: 5.691103166702708 [#---------] 6.28%

spotPython tuning: 5.691103166702708 [#---------] 7.75%
```

```
spotPython tuning: 5.691103166702708 [#---------] 9.21%

spotPython tuning: 5.691103166702708 [#---------] 11.65%

spotPython tuning: 5.691103166702708 [#---------] 14.02%

spotPython tuning: 5.691103166702708 [##--------] 16.29%

spotPython tuning: 5.691103166702708 [##--------] 18.68%

spotPython tuning: 5.691103166702708 [##--------] 21.00%

spotPython tuning: 5.691103166702708 [##--------] 23.57%

spotPython tuning: 5.691103166702708 [###-------] 26.20%

spotPython tuning: 5.691103166702708 [####------] 35.64%

spotPython tuning: 5.691103166702708 [#####-----] 49.05%

spotPython tuning: 5.691103166702708 [######----] 63.27%

spotPython tuning: 5.691103166702708 [########--] 77.54%

spotPython tuning: 5.691103166702708 [#########-] 91.58%

spotPython tuning: 5.691103166702708 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x287cffb50>
```

## 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")
```

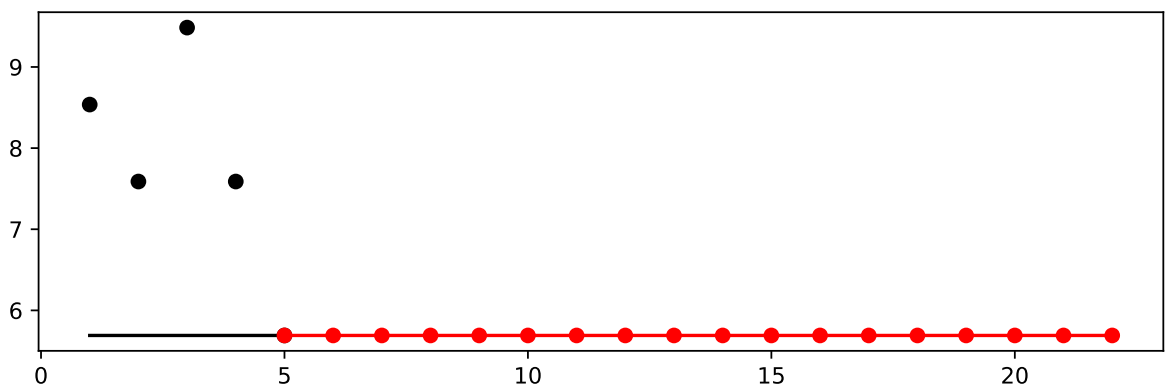



Figure 10.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- 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 |   3.6280771109650245 | 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 |                  0.0 | None
| tol         | float  | 0.001     |   0.001 |    0.01 | 0.006642600916881275 | None
| cache_size  | float  | 200.0     |   100.0 |   400.0 |   202.03372626175258 | 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
```

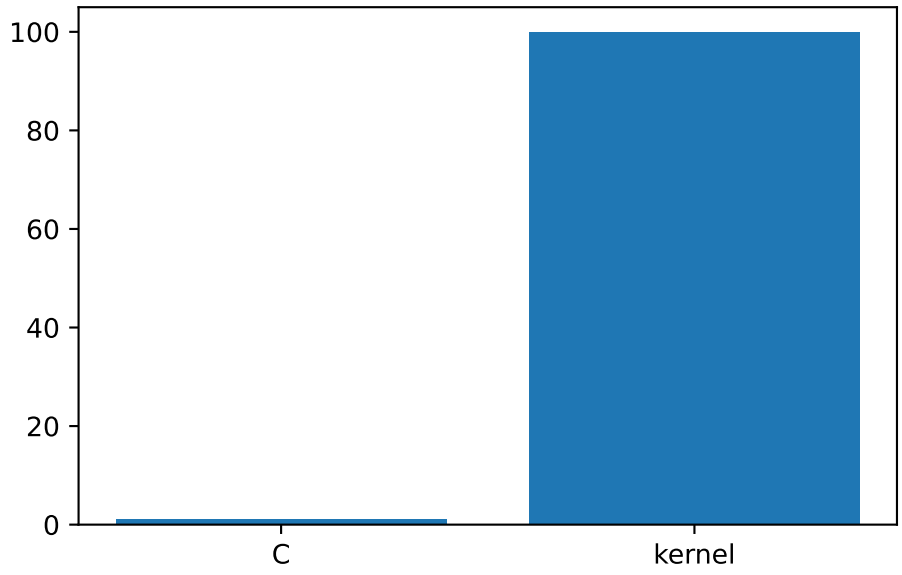



Figure 10.2: Variable importance plot, threshold 0.025.

### 10.9.2 Get Default Hyperparameters

```
from spotPython.hyperparameters.values import get_default_values, transform_hyper_paramete
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)
```

```
[[3.62807711e+00 1.00000000e+00 3.00000000e+00 0.00000000e+00
  0.00000000e+00 1.00000000e+00 0.00000000e+00 6.64260092e-03
  2.02033726e+02 1.00000000e+00]]
```

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

```
[{'C': 3.6280771109650245,
  'kernel': 'poly',
  'degree': 3,
  'gamma': 'scale',
  'coef0': 0.0,
  'shrinking': 1,
```

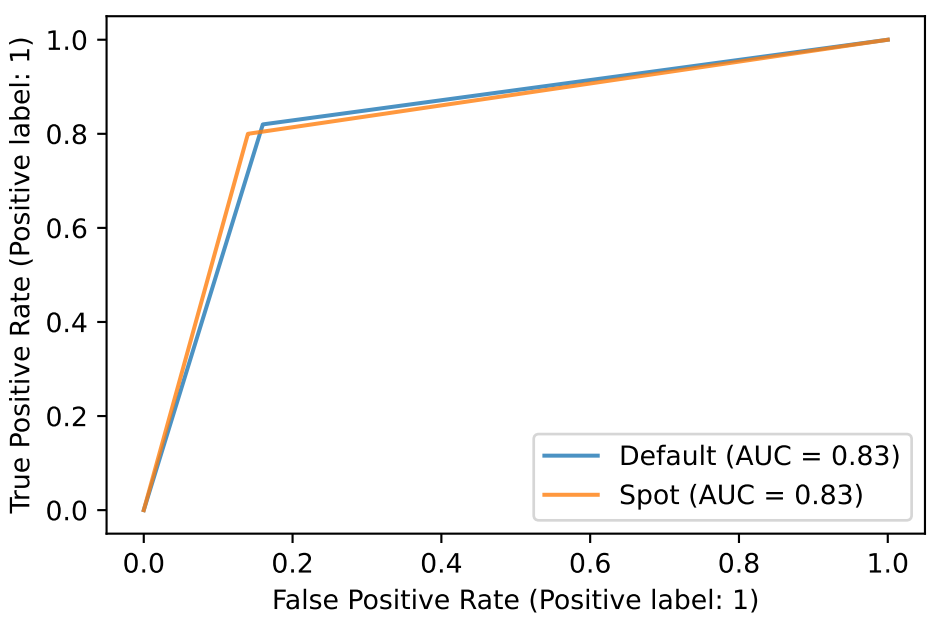
```
  'probability': 0,
  'tol': 0.006642600916881275,
  'cache_size': 202.03372626175258,
  'break_ties': 1}]
```

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

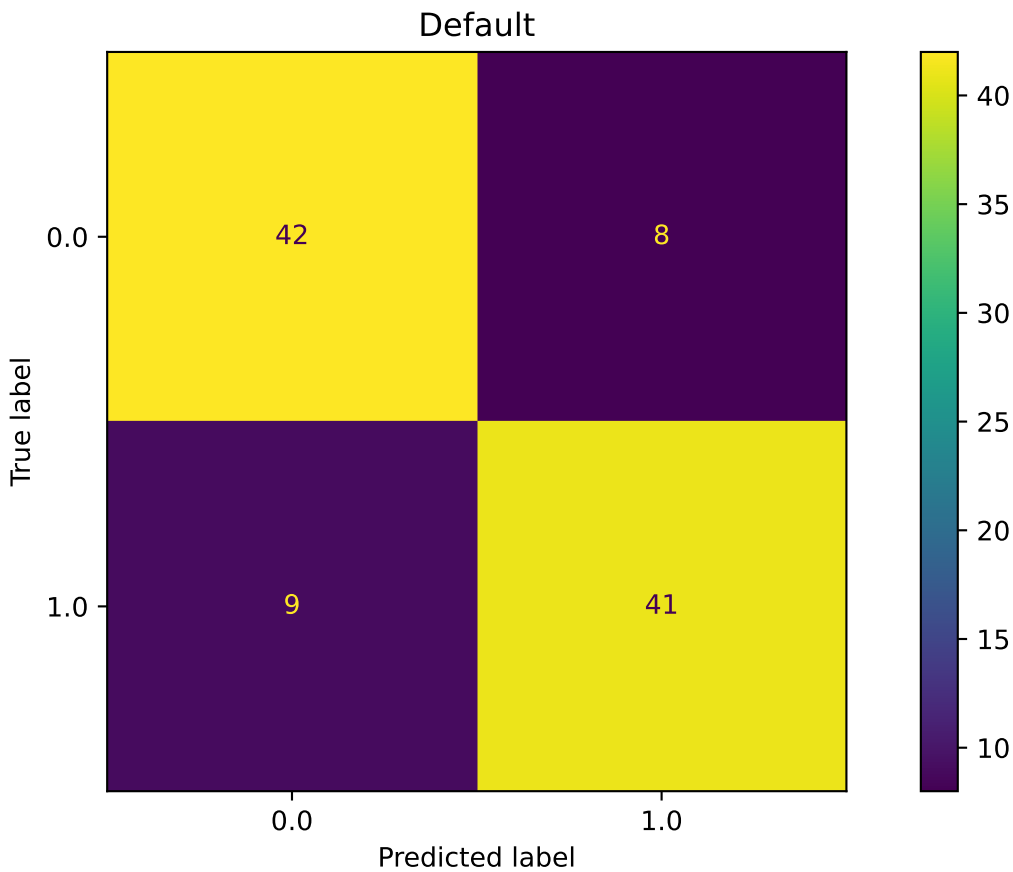
```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('svc',
                 SVC(C=3.6280771109650245, break_ties=1,
                     cache_size=202.03372626175258, kernel='poly',
                     probability=0, shrinking=1, tol=0.006642600916881275))])
```

### 10.9.4 Plot: Compare Predictions

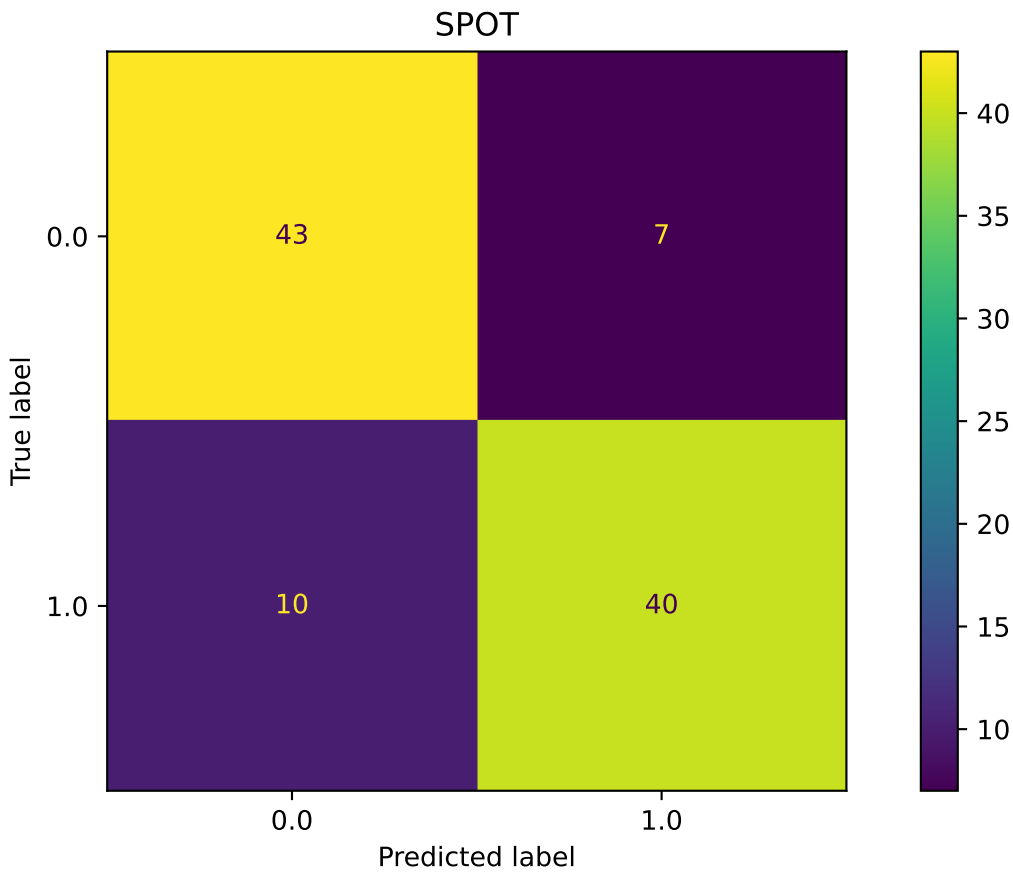
```python
from spotPython.plot.validation import plot_roc
plot_roc([model_default, model_spot], fun_control, model_names=["Default", "Spot"])
```



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




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



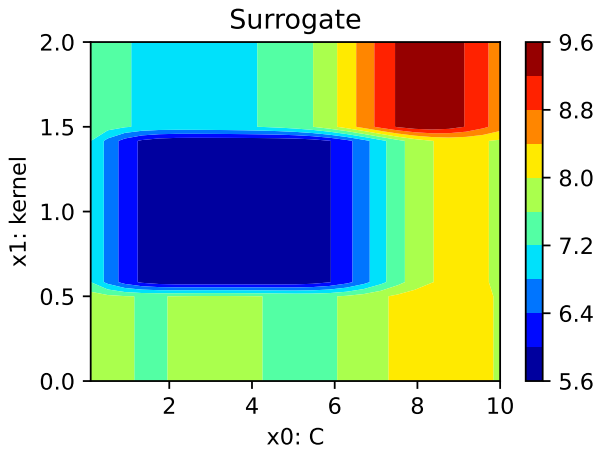

```
min(spot_tuner.y), max(spot_tuner.y)
```

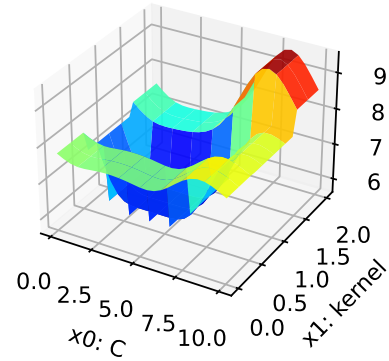
```
(5.691103166702708, 9.485171944504513)
```

### 10.9.5 Detailed Hyperparameter Plots

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

```
C:  1.2008877051836857
kernel:  99.99999999999999
```




### 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: 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.

## 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.
> - 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_p040025_1min_5init_2023-07-06_00-52-27
```

### 11.1.1 `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                  0.3.4
spotRiver                   0.0.93

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
```

## 11.2 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)
```

## 11.3 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 na
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})
```

## 11.4 Step 4: Specification of the Preprocessing Model

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

## 11.5 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
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}}
```

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

### 11.6.1 Modify hyperparameter of type factor

```
# modify_hyper_parameter_levels(fun_control, "leaf_model", ["LinearRegression"])
# fun_control["core_model_hyper_dict"]
```

### 11.6.2 Modify hyperparameter of type numeric and integer (boolean)

```
from spotPython.hyperparameters.values import modify_hyper_parameter_bounds
modify_hyper_parameter_bounds(fun_control, "delta", bounds=[1e-10, 1e-6])
# modify_hyper_parameter_bounds(fun_control, "min_samples_split", bounds=[3, 20])
modify_hyper_parameter_bounds(fun_control, "merit_preprune", [0, 0])
```

## 11.7 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
                })
```

## 11.8 Step 8: Calling the SPOT Function

### 11.8.1 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)
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_powe
| 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
```

### 11.8.2 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
X_start = get_default_hyperparameters_as_array(fun_control)
```

```
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,
                   max_time = MAX_TIME,
                   tolerance_x = np.sqrt(np.spacing(1)),
                   var_type = var_type,
                   var_name = var_name,
                   show_progress= True,
                   fun_control = fun_control,
                   design_control={"init_size": INIT_SIZE},
                   surrogate_control={"noise": True,
                                      "cod_type": "norm",
```

```
                                        "min_theta": -4,
                                        "max_theta": 3,
                                        "n_theta": len(var_name),
                                        "model_fun_evals": 10_000
                                        })
spot_tuner.run(X_start=X_start)
```

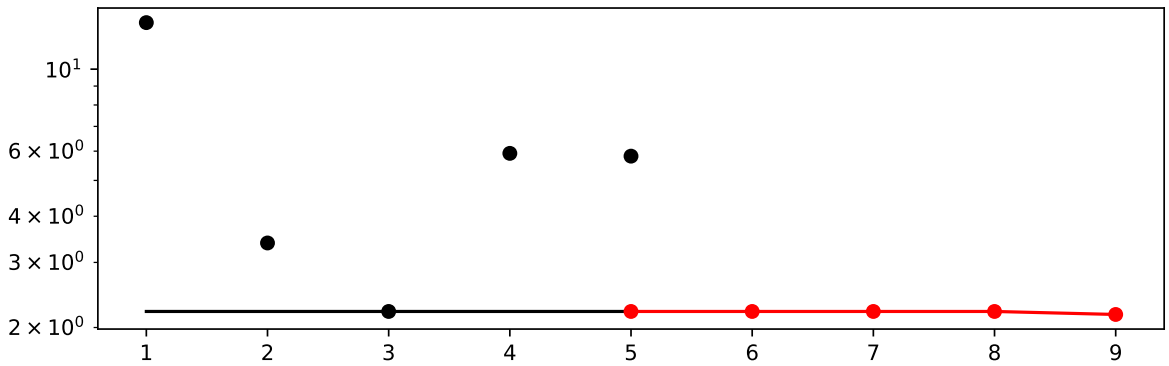
```
spotPython tuning: 2.210327218055443 [###-------] 25.57%

spotPython tuning: 2.210327218055443 [######----] 57.69%

spotPython tuning: 2.210327218055443 [########--] 78.33%

spotPython tuning: 2.210327218055443 [#########-] 94.91%

spotPython tuning: 2.169306422146142 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x2c191eec0>
```

## 11.9 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.pd
```



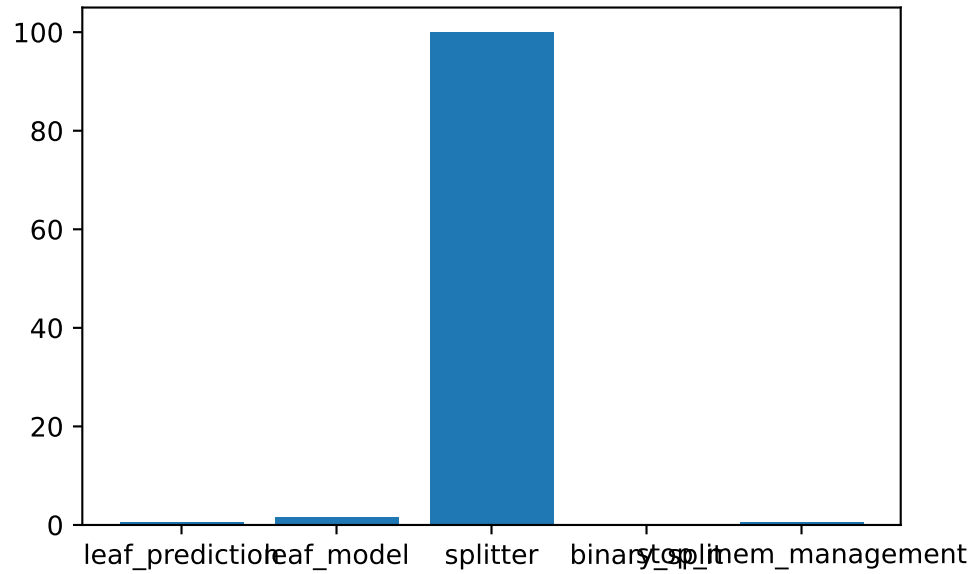

- Print the Results

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

```
| name                    | type   | default          |    lower |     upper |               tu
|-------------------------|--------|------------------|----------|-----------|----------------
| grace_period            | int    | 200              |     10.0 |    1000.0 |               85
| max_depth               | int    | 20               |      2.0 |      20.0 |
| delta                   | float  | 1e-07            |    1e-10 |     1e-06 |               1e
| tau                     | float  | 0.05             |     0.01 |       0.1 |
| 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 |
| 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 |               12
| switch_significance     | float  | 0.05             |     0.01 |       0.1 |                0
| binary_split            | factor | 0                |      0.0 |       1.0 |
| max_size                | float  | 500.0            |    100.0 |    1000.0 | 106.98158548708
| memory_estimate_period  | int    | 1000000          | 100000.0 | 1000000.0 |            84176
| 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 |
```

### 11.9.1 Show variable importance

```
spot_tuner.plot_importance(threshold=0.0025, filename="./figures/" + experiment_name+"_imp
```




### 11.9.2 Build and Evaluate HTR Model with Tuned Hyperparameters

```
m = test.shape[0]
a = int(m/2)-50
b = int(m/2)
```

### 11.9.3 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 na
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})
```

### 11.9.4 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
from spotPython.hyperparameters.values import get_default_hyperparameters_as_array
X_start = get_default_hyperparameters_as_array(fun_control)
```

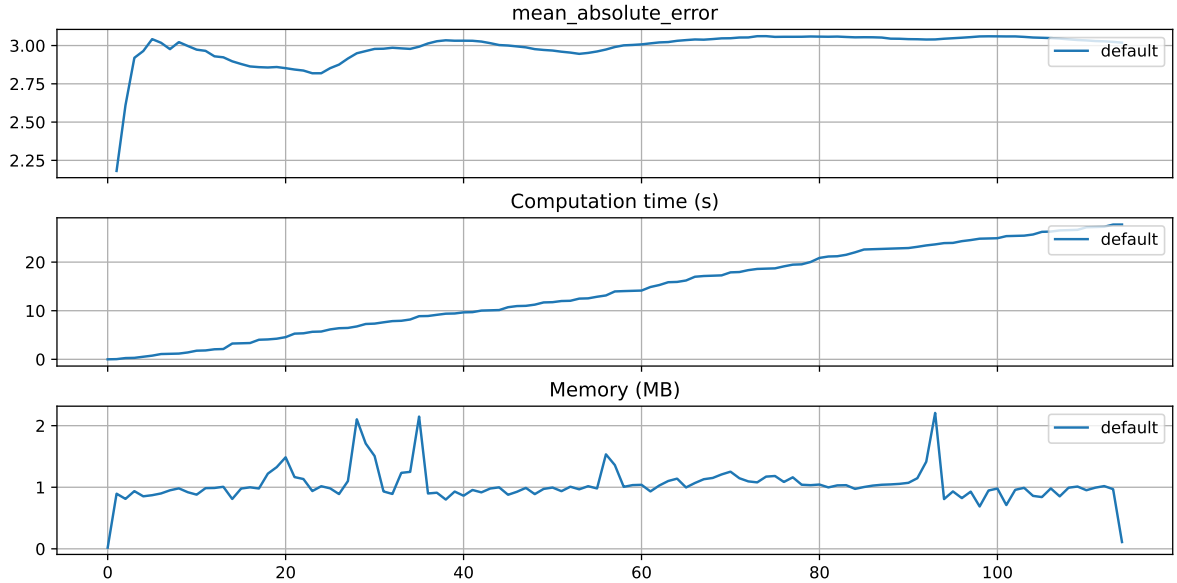
```
model_default = get_one_core_model_from_X(X_start, fun_control)
model_default
```

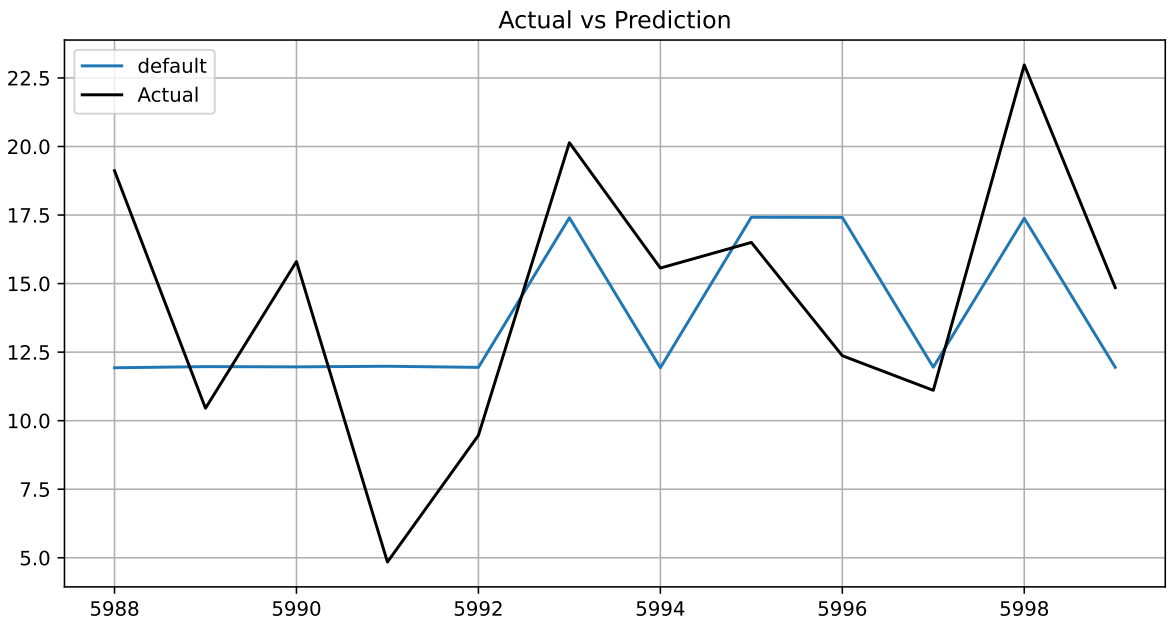
```
HoeffdingAdaptiveTreeRegressor (
  grace_period=200
  max_depth=1048576
  delta=1e-07
  tau=0.05
  leaf_prediction="mean"
  leaf_model=LinearRegression (
    optimizer=SGD (
      lr=Constant (
        learning_rate=0.01
      )
    )
    loss=Squared ()
    l2=0.
    l1=0.
    intercept_init=0.
    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_horiz
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_co
```






### 11.9.5 Get SPOT Results

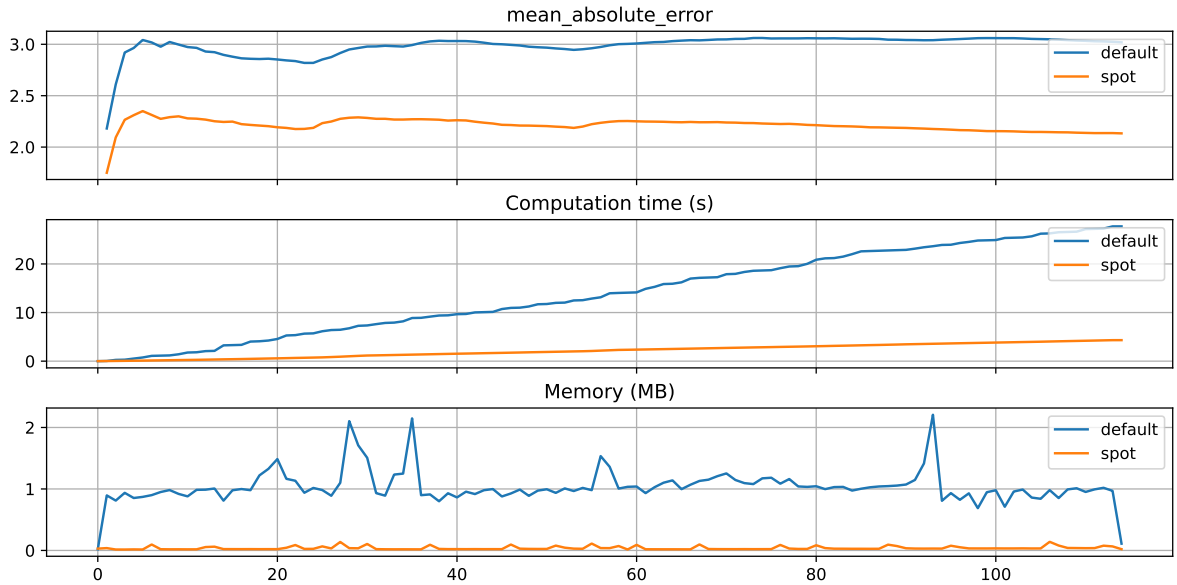
```
from spotPython.hyperparameters.values import get_one_core_model_from_X
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
model_spot = get_one_core_model_from_X(X, fun_control)
model_spot
```

```
HoeffdingAdaptiveTreeRegressor (
  grace_period=857
  max_depth=128
  delta=1e-06
  tau=0.1
  leaf_prediction="adaptive"
  leaf_model=LinearRegression (
    optimizer=SGD (
      lr=Constant (
        learning_rate=0.01
      )
    )
    loss=Squared ()
    l2=0.
    l1=0.
```

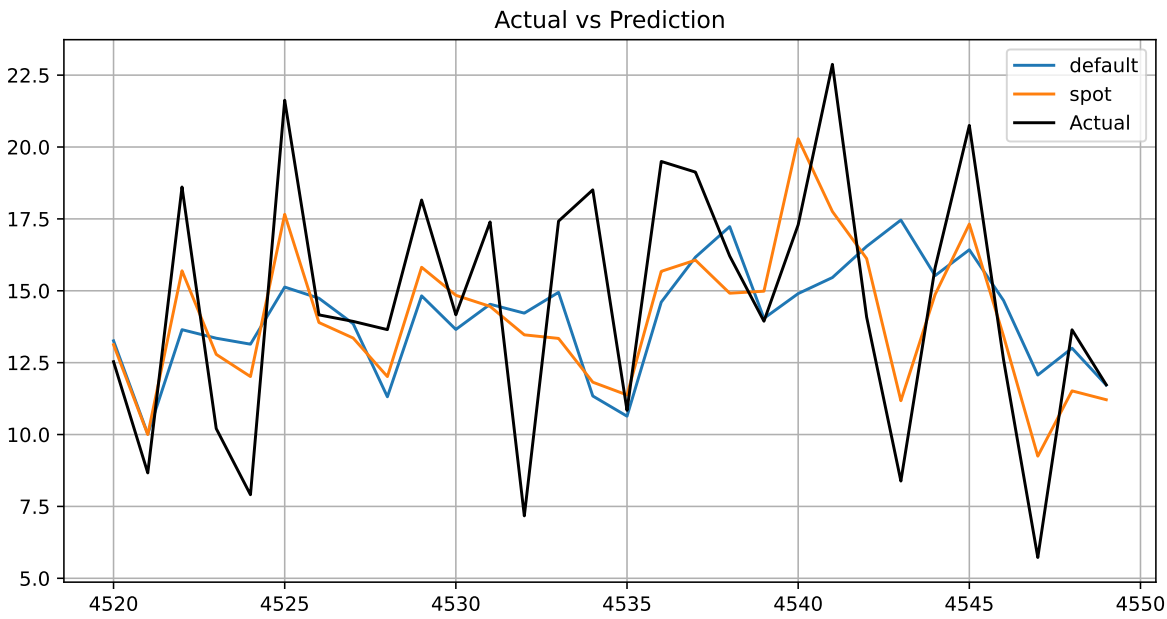
```
    intercept_init=0.
    intercept_lr=Constant (
      learning_rate=0.01
    )
    clip_gradient=1e+12
    initializer=Zeros ()
  )
  model_selector_decay=0.9
  nominal_attributes=None
  splitter=QOSplitter (
    radius=0.25
    allow_multiway_splits=False
  )
  min_samples_split=4
  bootstrap_sampling=0
  drift_window_threshold=125
  drift_detector=ADWIN (
    delta=0.002
    clock=32
    max_buckets=5
    min_window_length=5
    grace_period=10
  )
  switch_significance=0.01
  binary_split=0
  max_size=106.981585
  memory_estimate_period=841767
  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"],
                )
```

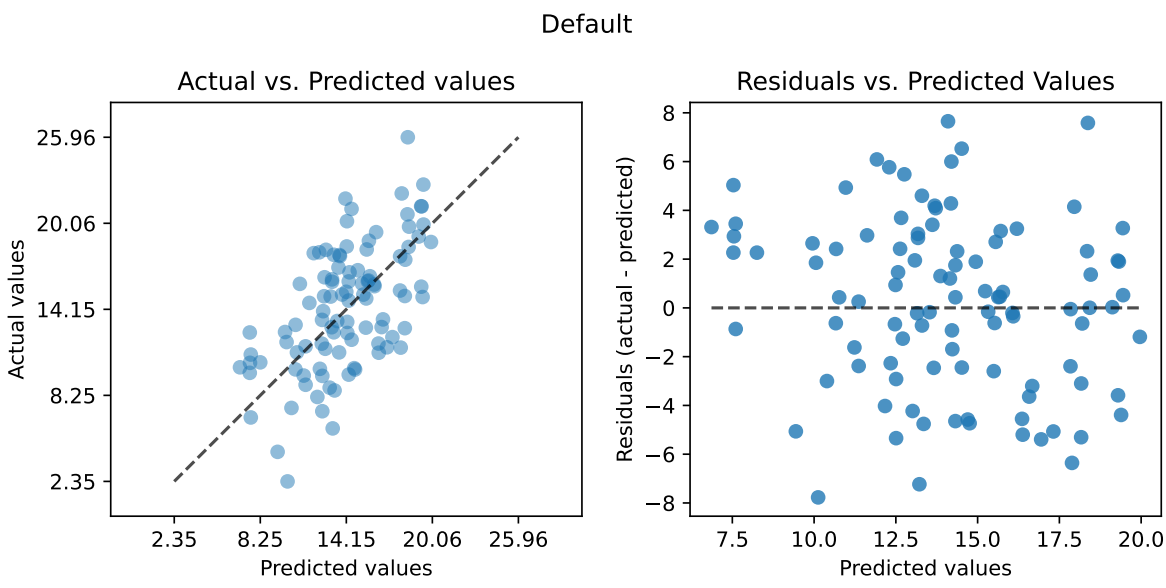
```
df_labels=["default", "spot"]
plot_bml_oml_horizon_metrics(df_eval = [df_eval_default, df_eval_spot], log_y=False, df_la
```

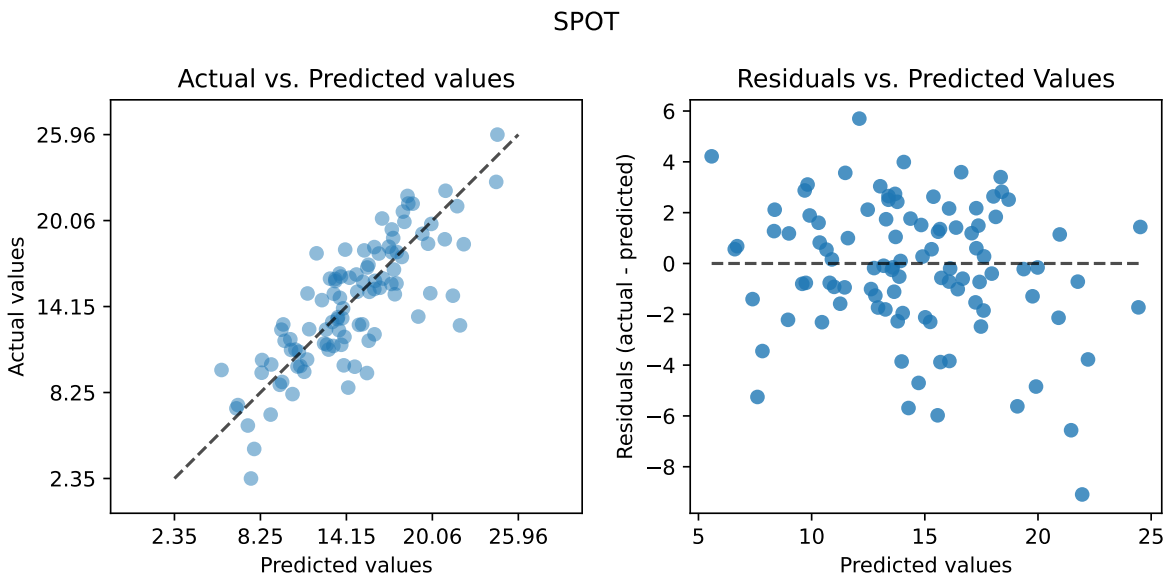



```
a = int(m/2)+20
b = int(m/2)+50
plot_bml_oml_horizon_predictions(df_true = [df_true_default[a:b], df_true_spot[a:b]], targ
```




```
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
```






### 11.9.6 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}
```

### 11.9.7 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': 19,
 'n_branches': 9,
 'n_leaves': 10,
 'n_active_leaves': 30,
 'n_inactive_leaves': 0,
 'height': 6,
 'total_observed_weight': 39002.0,
 'n_alternate_trees': 15,
 'n_pruned_alternate_trees': 7,
 'n_switch_alternate_trees': 1}
```

```
from spotPython.utils.eda import compare_two_tree_models
print(compare_two_tree_models(model_default, model_spot))
```

```
| Parameter                |   Default |   Spot |
|--------------------------|-----------|--------|
| n_nodes                  |        35 |     19 |
| n_branches               |        17 |      9 |
| n_leaves                 |        18 |     10 |
```

```
| n_active_leaves          |        96 |     30 |
| n_inactive_leaves        |         0 |      0 |
| height                   |         6 |      6 |
| total_observed_weight    |     39002 |  39002 |
| n_alternate_trees        |        21 |     15 |
| n_pruned_alternate_trees |         6 |      7 |
| n_switch_alternate_trees |         2 |      1 |
```

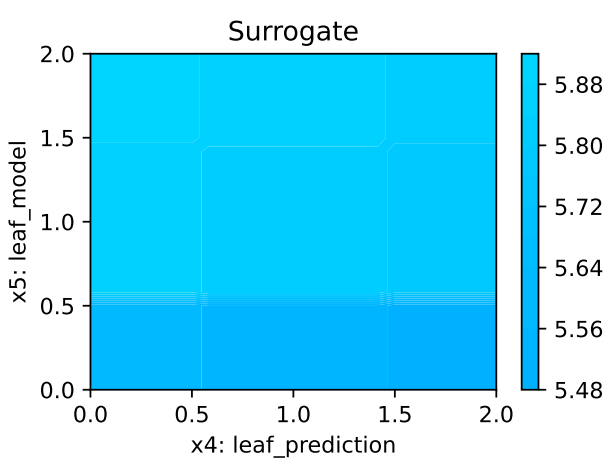
```
min(spot_tuner.y), max(spot_tuner.y)
```

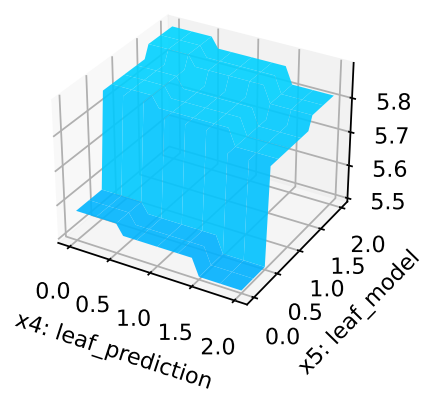
```
(2.169306422146142, 13.363060338886816)
```

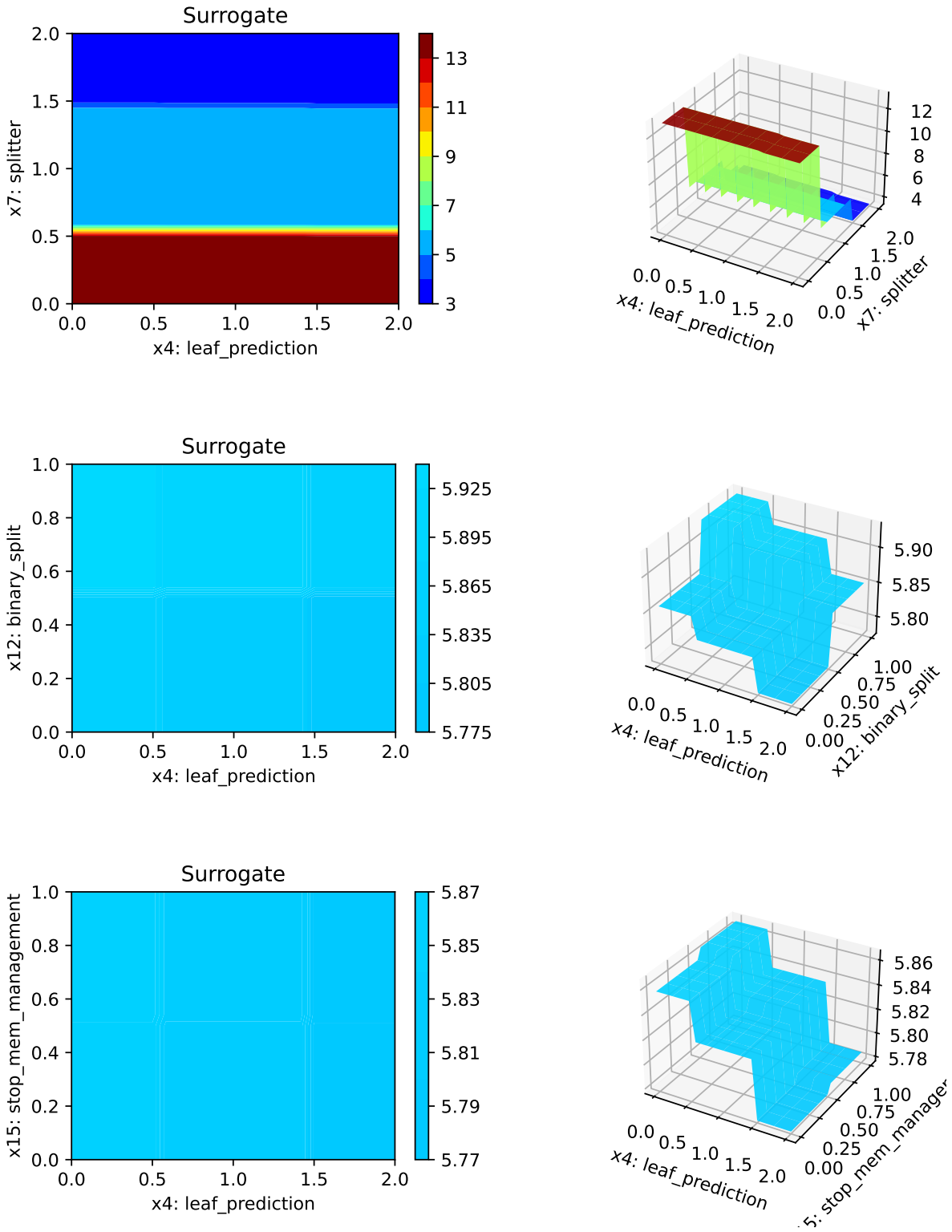
### 11.9.8 Detailed Hyperparameter Plots

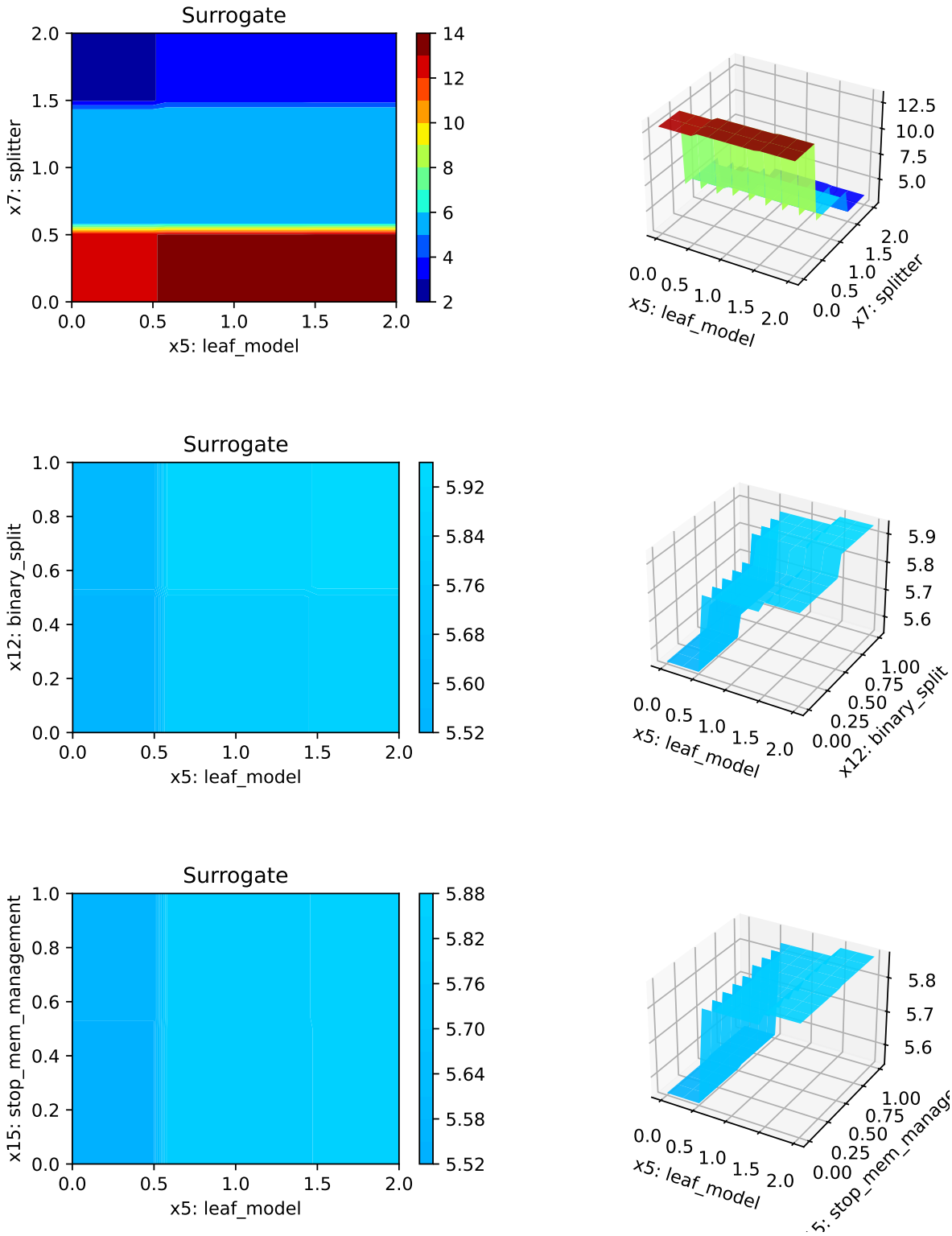
```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

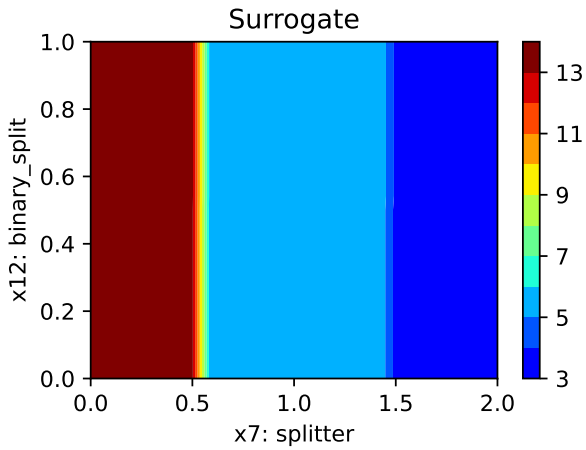
```
leaf_prediction:  0.4458129635043254
leaf_model:  1.4317508451585736
splitter:  100.00000000000001
binary_split:  0.052573666489981026
stop_mem_management:  0.433345294478879
```

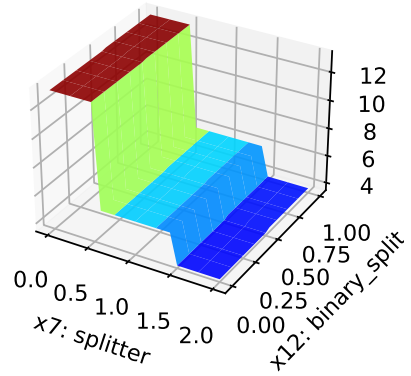


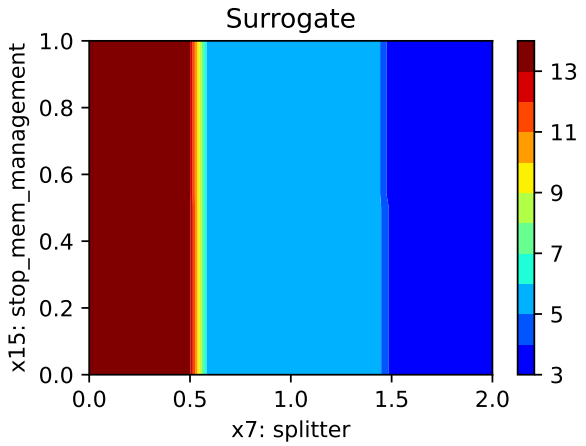



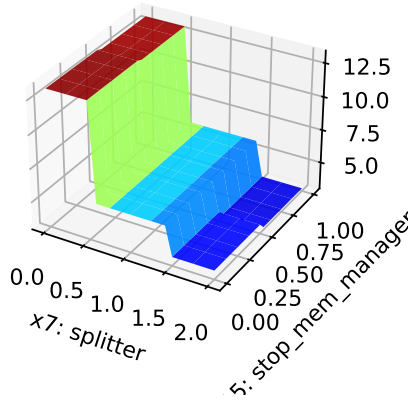

Surrogate
x7: splitter
x4: leaf_prediction
Surrogate
x12: binary_split
x4: leaf_prediction
Surrogate
x15: stop_mem_management
x4: leaf_prediction

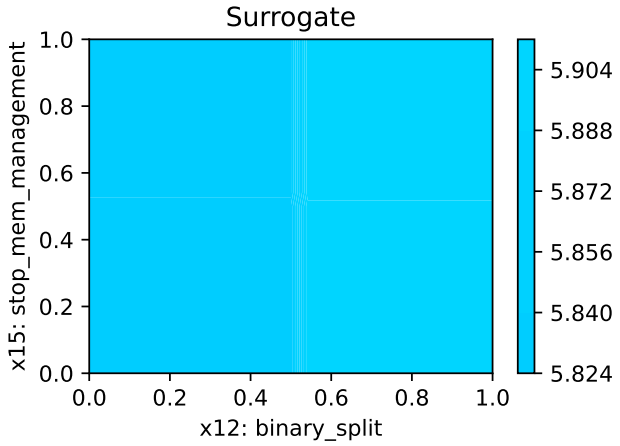

Surrogate
x7: splitter
x5: leaf_model
Surrogate
x12: binary_split
x5: leaf_model
Surrogate
x15: stop_mem_management
x5: leaf_model
x5: leaf_model
x7: splitter
x5: leaf_model
x12: binary_split
x5: leaf_model

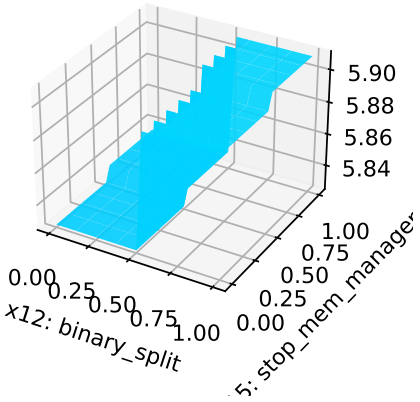

Surrogate
x12: binary_split
x7: splitter
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0
13
11
9
7
5
3


12
10
8
6
4
0.0
0.5
1.0
1.5
2.0
x7: splitter
1.00
0.75
0.50
0.25
0.00
x12: binary_split


Surrogate
x15: stop_mem_management
x7: splitter
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0
13
11
9
7
5
3


12.5
10.0
7.5
5.0
0.0
0.5
1.0
1.5
2.0
x7: splitter
1.00
0.75
0.50
0.25
0.00


Surrogate
x15: stop_mem_management
x12: binary_split
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
5.904
5.888
5.872
5.856
5.840
5.824


5.90
5.88
5.86
5.84
0.00
0.25
0.50
0.75
1.00
x12: binary_split
1.00
0.75
0.50
0.25
0.00

### 11.9.9 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
```

### 11.9.10 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)
```

# 12 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.

`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
```

Results that refer to the `Ray Tune` package are taken from https://PyTorch.org/tutorials/beginner/hyperparameter_tuning_tutorial.html[2].

## 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

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

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

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 `"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.

```
MAX_TIME = 1
INIT_SIZE = 5
DEVICE = "auto" # "cpu"
PREFIX = "14-torch"
```

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

```
mps
```

```
import os
import copy
import socket
import warnings
from datetime import datetime
from dateutil.tz import tzlocal
from spotPython.utils.file import get_experiment_name
experiment_name = get_experiment_name(prefix=PREFIX)
print(experiment_name)
if not os.path.exists('./figures'):
    os.makedirs('./figures')
warnings.filterwarnings("ignore")
```

```
14-torch_p040025_2023-07-06_01-29-12
```

## 12.2 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,)
```

## 12.3 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
```

## 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. 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})
```

## 12.5 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 12.5.0.1 first, before the extended implementation with `spotPython` is shown in Section 12.5.0.2.

#### 12.5.0.1 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 `l1` and `l2`, 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)
```

#### 12.5.0.2 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
```

### 12.5.1 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
```

### 12.5.2 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
add_core_model_to_fun_control(core_model=core_model,
                              fun_control=fun_control,
                              hyper_dict=TorchHyperDict,
                              filename=None)
```

### 12.5.3 The Search Space: Hyperparameters

In Section 12.5.4, 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.

### 12.5.4 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.

### 12.5.5 Configuring the Search Space With spotPython

#### 12.5.5.1 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': ['Adadelta',
   '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`

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.

#### 12.6.0.1 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.

#### 12.6.0.2 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
modify_hyper_parameter_bounds(fun_control,
    "batch_size", bounds=[1, 5])
modify_hyper_parameter_bounds(fun_control,
    "k_folds", bounds=[0, 0])
modify_hyper_parameter_bounds(fun_control,
    "patience", bounds=[3, 3])
```

#### 12.6.0.3 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
modify_hyper_parameter_levels(fun_control,
     "optimizer", ["SGD", "Adam"])
```

The hyperparameter `optimizer` can be de-activated by choosing only one value (level), here: `"SGD"`.

```
modify_hyper_parameter_levels(fun_control, "optimizer", ["SGD"])
```

As discussed in Section 12.6.1, 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:

```
modify_hyper_parameter_levels(fun_control, "optimizer",
    ["Adadelta", "Adagrad", "Adam", "AdamW", "Adamax", "ASGD",
    "NAdam", "RAdam", "RMSprop", "SGD"])
```

### 12.6.1 Optimizers

Table 12.1 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 12.1: 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* | - | - | - | - | - |
| ASGD | 1e-2 | .9 | 0. | - | F | - | - | - | - | 1e-4 | .75 | - | - | - |
| LBFGS | 1. | - | - | - | - | - | - | - | - | - | - | - | - | - |
| NAdam | 2e-3 | - | 0. | - | - | - | - | - | *a* | - | - | 0 | - | - |
| RAdam | 1e-3 | - | 0. | - | - | - | - | - | *a* | - | - | - | - | - |
| RMSprop | 1e-2 | 0. | 0. | - | - | - | - | - | *a* | - | - | - | - | - |
| Rprop | 1e-2 | - | - | - | - | - | - | - | - | - | *b* | *c* | - | - |
| SGD | *R* | 0. | 0. | 0. | F | - | - | - | - | - | - | - | - | - |

`spotPython` implements an `optimization` handler that maps the optimizer names to the corresponding `PyTorch` optimizers.

**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.

**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`.

```
modify_hyper_parameter_bounds(fun_control,
    "lr_mult", bounds=[1.0, 1.0])
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: 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.

### 12.7.2 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).

### 12.7.3 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.

### 12.7.4 Overview of the Evaluation Settings

#### 12.7.4.1 Settings for the Hyperparameter Tuning

An overview of the training evaluations is shown in Table 12.2. `"train_cv"` and `"test_cv"` use `sklearn.model_selection.KFold()` internally. More details on the data splitting are provided in Section 18.14 (in the Appendix).

Table 12.2: 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. |

#### 12.7.4.2 Settings for the Final Evaluation of the Tuned Architecture

##### 12.7.4.2.1 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 `train`data 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.

##### 12.7.4.2.2 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 18.14.1.4 describes the final evaluation of the tuned architecture.

```
fun_control.update({
    "eval": "train_hold_out",
    "path": "torch_model.pt",
    "shuffle": True})
```

### 12.7.5 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 flexibilty, 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["de
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`.

```
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)

lower = get_bound_values(fun_control, "lower")
upper = get_bound_values(fun_control, "upper")
```

Now, the dictionary `fun_control` contains all information needed for the hyperparameter tuning. Before the hyperparameter tuning is started, it is recommended to take a look at the experimental design. The method `gen_design_table` generates a design table as follows:

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

```
| name         | type   | default   |   lower |   upper | transform             |
|--------------|--------|-----------|---------|---------|-----------------------|
| l1           | int    | 5         |     2   |     9   | transform_power_2_int |
| l2           | int    | 5         |     2   |     9   | transform_power_2_int |
| lr_mult      | float  | 1.0       |     1   |     1   | None                  |
| batch_size   | int    | 4         |     1   |     5   | transform_power_2_int |
| epochs       | int    | 3         |     3   |     4   | transform_power_2_int |
| k_folds      | int    | 1         |     0   |     0   | None                  |
| patience     | int    | 5         |     3   |     3   | None                  |
| optimizer    | factor | SGD       |     0   |     9   | None                  |
| sgd_momentum | float  | 0.0       |     0.9 |     0.9 | None                  |
```

This allows to check if all information is available and if the information is correct. **?@tbl-design** shows the experimental design for the hyperparameter tuning. The table shows the hyperparameters, their types, default values, lower and upper bounds, and the transformation function. The transformation function is used to transform the hyperparameter values from the unit hypercube to the original domain. The transformation function is applied to the hyperparameter values before the evaluation of the objective function. Hyperparameter transformations are shown in the column "transform", e.g., the `l1` default is `5`, which results in the value $2^5 = 32$ for the network, because the transformation `transform_power_2_int` was selected in the `JSON` file. The default value of the `batch_size` is set to `4`, which results in a batch size of $2^4 = 16$.

### 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 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
X_start = get_default_hyperparameters_as_array(fun_control)
```

### 12.8.4 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 12.9.

```
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,
                   max_time = MAX_TIME,
                   tolerance_x = np.sqrt(np.spacing(1)),
                   var_type = var_type,
                   var_name = var_name,
                   show_progress= True,
                   fun_control = fun_control,
                   design_control={"init_size": INIT_SIZE},
                   surrogate_control={"noise": True,
                                      "cod_type": "norm",
                                      "min_theta": -4,
                                      "max_theta": 3,
                                      "n_theta": len(var_name),
                                      "model_fun_evals": 10_000
                                      })
spot_tuner.run(X_start=X_start)
```

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

MulticlassAccuracy: 0.3889499902725220 | Loss: 1.6403590366363525 | Acc: 0.3889500000000000.
Epoch: 2 |
```

```
MulticlassAccuracy: 0.4578999876976013 | Loss: 1.4816969134330749 | Acc: 0.4579000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.4945999979972839 | Loss: 1.3767625138282775 | Acc: 0.4946000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5118499994277954 | Loss: 1.3446329971313478 | Acc: 0.5118500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5447499752044678 | Loss: 1.2767737101554870 | Acc: 0.5447500000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5664499998092651 | Loss: 1.2234437763214112 | Acc: 0.5664500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5648499727249146 | Loss: 1.2325385323524476 | Acc: 0.5648500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5896499752998352 | Loss: 1.1611093239784240 | Acc: 0.5896500000000000.
Epoch: 9 |

MulticlassAccuracy: 0.6015999913215637 | Loss: 1.1370150957107543 | Acc: 0.6016000000000000.
Epoch: 10 |

MulticlassAccuracy: 0.6074000000953674 | Loss: 1.1378371593475343 | Acc: 0.6074000000000001.
Epoch: 11 |

MulticlassAccuracy: 0.6036999821662903 | Loss: 1.1592556796073914 | Acc: 0.6037000000000000.
Epoch: 12 |

MulticlassAccuracy: 0.5997499823570251 | Loss: 1.1987680685997009 | Acc: 0.5997500000000000.
Early stopping at epoch 11
Returned to Spot: Validation loss: 1.1987680685997009

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

MulticlassAccuracy: 0.3920499980449677 | Loss: 1.6102165319681168 | Acc: 0.3920500000000000.
Epoch: 2 |
```

```
MulticlassAccuracy: 0.4390000104904175 | Loss: 1.5077767979741097 | Acc: 0.4390000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.4700999855995178 | Loss: 1.4581756867766380 | Acc: 0.4701000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.4981499910354614 | Loss: 1.3969129746913911 | Acc: 0.4981500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5059000253677368 | Loss: 1.3693460956692696 | Acc: 0.5059000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5133500099182129 | Loss: 1.3540988440275192 | Acc: 0.5133500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5081499814987183 | Loss: 1.3817692994177342 | Acc: 0.5081500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5159500241279602 | Loss: 1.3653468480706215 | Acc: 0.5159500000000000.
Returned to Spot: Validation loss: 1.3653468480706215

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

MulticlassAccuracy: 0.0958499982953072 | Loss: 2.3086834851264952 | Acc: 0.0958500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.0987000018358231 | Loss: 2.3107500833988190 | Acc: 0.0987000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.0958499982953072 | Loss: 2.3054559610605239 | Acc: 0.0958500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.1013000011444092 | Loss: 2.3091404678583145 | Acc: 0.1013000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.0958499982953072 | Loss: 2.3109533527135850 | Acc: 0.0958500000000000.
Epoch: 6 |
```

```
MulticlassAccuracy: 0.0987000018358231 | Loss: 2.3080133529186249 | Acc: 0.0987000000000000.
Early stopping at epoch 5
Returned to Spot: Validation loss: 2.308013352918625

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

MulticlassAccuracy: 0.3910000026226044 | Loss: 1.6194829273104667 | Acc: 0.3910000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.4532499909400940 | Loss: 1.5181912495672703 | Acc: 0.4532500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.5023999810218811 | Loss: 1.3594324642419815 | Acc: 0.5024000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5066999793052673 | Loss: 1.3639220094040037 | Acc: 0.5067000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5313000082969666 | Loss: 1.3084210138827563 | Acc: 0.5313000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5376499891281128 | Loss: 1.3020537653062492 | Acc: 0.5376500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5404999852180481 | Loss: 1.2979997927054763 | Acc: 0.5405000000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5505999922752380 | Loss: 1.2794678398683668 | Acc: 0.5506000000000000.
Returned to Spot: Validation loss: 1.2794678398683668

config: {'l1': 64, 'l2': 512, 'lr_mult': 1.0, 'batch_size': 16, 'epochs': 16, 'k_folds': 0,
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan

config: {'l1': 512, 'l2': 256, 'lr_mult': 1.0, 'batch_size': 16, 'epochs': 8, 'k_folds': 0,
Epoch: 1 |

MulticlassAccuracy: 0.5067499876022339 | Loss: 1.3663547940254210 | Acc: 0.5067500000000000.
Epoch: 2 |
```

```
MulticlassAccuracy: 0.5563499927520752 | Loss: 1.2667929081201554 | Acc: 0.5563500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.5666499733924866 | Loss: 1.2227724067926407 | Acc: 0.5666500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.6025000214576721 | Loss: 1.1719128470897675 | Acc: 0.6025000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5952500104904175 | Loss: 1.2412489697217941 | Acc: 0.5952499999999999.
Epoch: 6 |

MulticlassAccuracy: 0.5884500145912170 | Loss: 1.2785740818262101 | Acc: 0.5884500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.6009500026702881 | Loss: 1.3023499223232269 | Acc: 0.6009500000000000.
Early stopping at epoch 6
Returned to Spot: Validation loss: 1.3023499223232269

spotPython tuning: 1.1987680685997009 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x283a47550>
```

## 12.9 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard.

### 12.9.1 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.

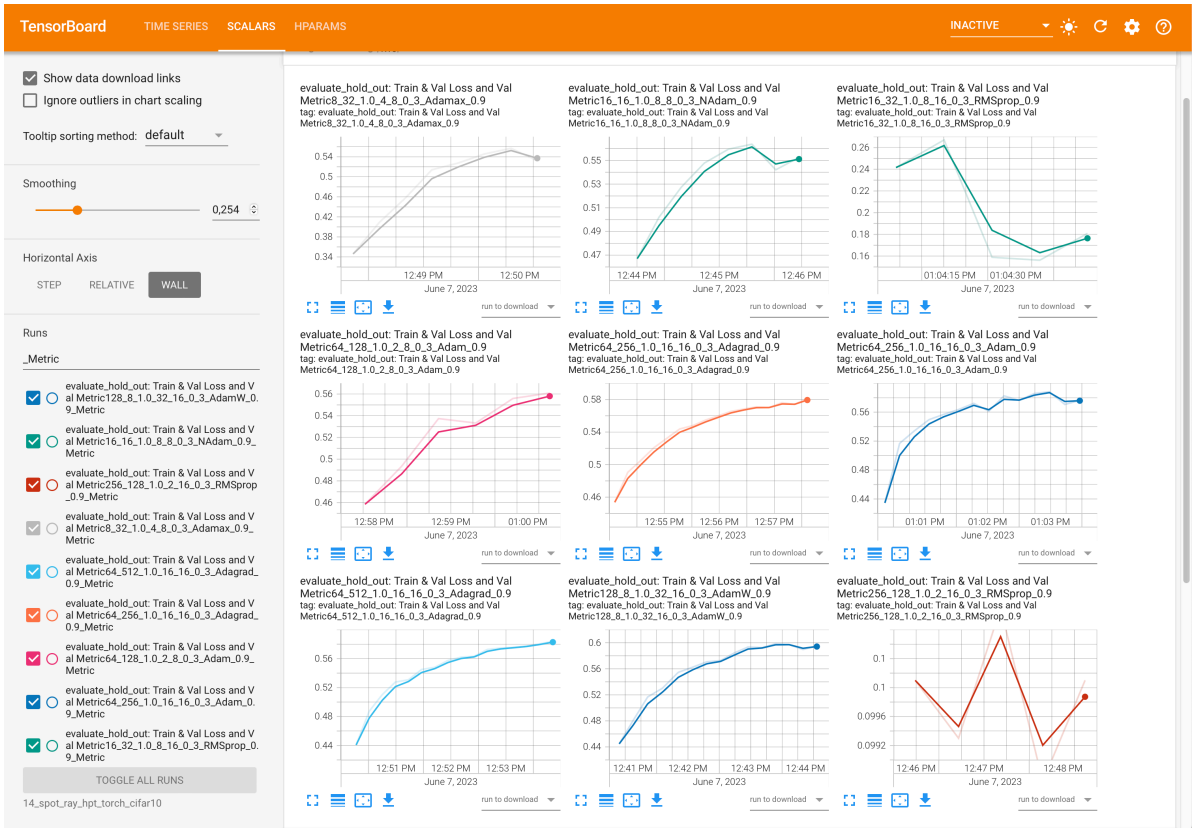
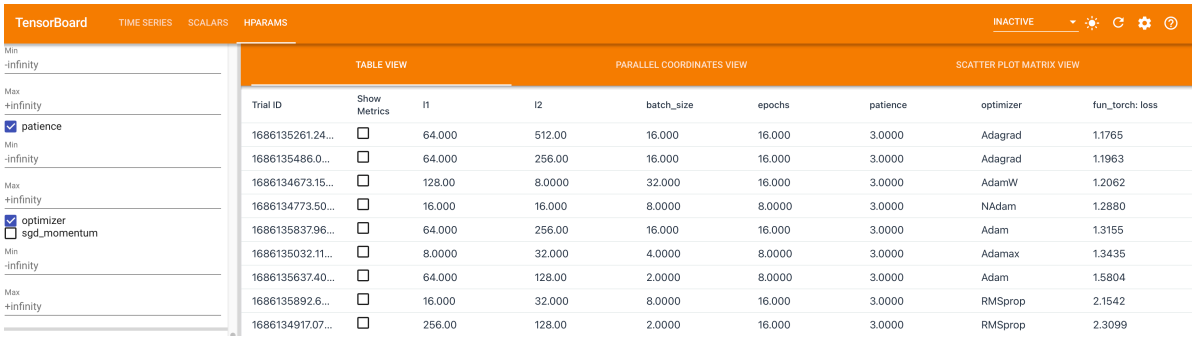



Figure 12.1: Tensorboard




| Trial ID | Show Metrics | l1 | l2 | batch_size | epochs | patience | optimizer | fun_torch: loss |
|---|---|---|---|---|---|---|---|---|
| 1686135261.24... | ☐ | 64.000 | 512.00 | 16.000 | 16.000 | 3.0000 | Adagrad | 1.1765 |
| 1686135486.0... | ☐ | 64.000 | 256.00 | 16.000 | 16.000 | 3.0000 | Adagrad | 1.1963 |
| 1686134673.15... | ☐ | 128.00 | 8.0000 | 32.000 | 16.000 | 3.0000 | AdamW | 1.2062 |
| 1686134773.50... | ☐ | 16.000 | 16.000 | 8.0000 | 8.0000 | 3.0000 | NAdam | 1.2880 |
| 1686135837.96... | ☐ | 64.000 | 256.00 | 16.000 | 16.000 | 3.0000 | Adam | 1.3155 |
| 1686135032.11... | ☐ | 8.0000 | 32.000 | 4.0000 | 8.0000 | 3.0000 | Adamax | 1.3435 |
| 1686135637.40... | ☐ | 64.000 | 128.00 | 2.0000 | 8.0000 | 3.0000 | Adam | 1.5804 |
| 1686135892.6... | ☐ | 16.000 | 32.000 | 8.0000 | 16.000 | 3.0000 | RMSprop | 2.1542 |
| 1686134917.07... | ☐ | 256.00 | 128.00 | 2.0000 | 16.000 | 3.0000 | RMSprop | 2.3099 |

Figure 12.2: Tensorboard

### 12.9.2 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)
```

## 12.10 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")
```

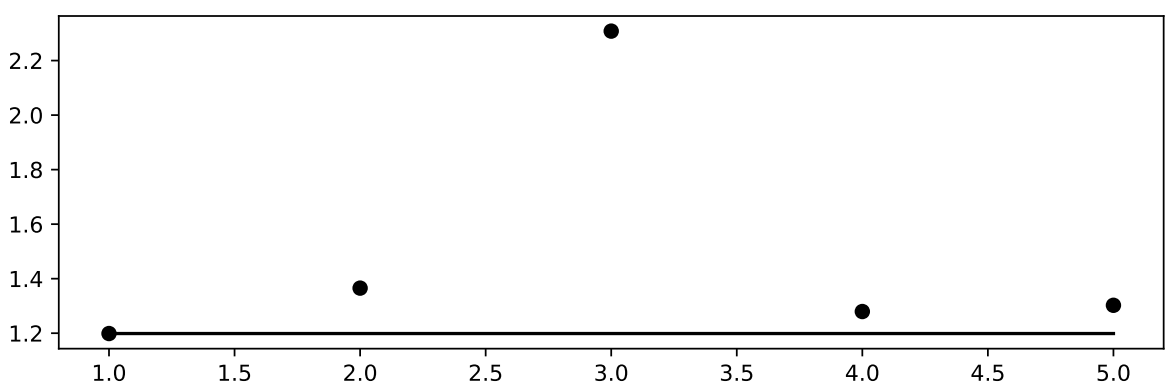



Figure 12.3: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

**?@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 |     7.0 | transform_power_2_int |
| l2           | int    | 5         |     2.0 |     9.0 |     3.0 | transform_power_2_int |
| lr_mult      | float  | 1.0       |     1.0 |     1.0 |     1.0 | None                  |
| batch_size   | int    | 4         |     1.0 |     5.0 |     5.0 | transform_power_2_int |
| epochs       | int    | 3         |     3.0 |     4.0 |     4.0 | transform_power_2_int |
| k_folds      | int    | 1         |     0.0 |     0.0 |     0.0 | None                  |
| patience     | int    | 5         |     3.0 |     3.0 |     3.0 | None                  |
| optimizer    | factor | SGD       |     0.0 |     9.0 |     3.0 | None                  |
| sgd_momentum | float  | 0.0       |     0.9 |     0.9 |     0.9 | None                  |
```

To visualize the most important hyperparameters, `spotPython` provides the function `plot_importance`. The following code generates the importance plot from **?@fig-importance**.

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

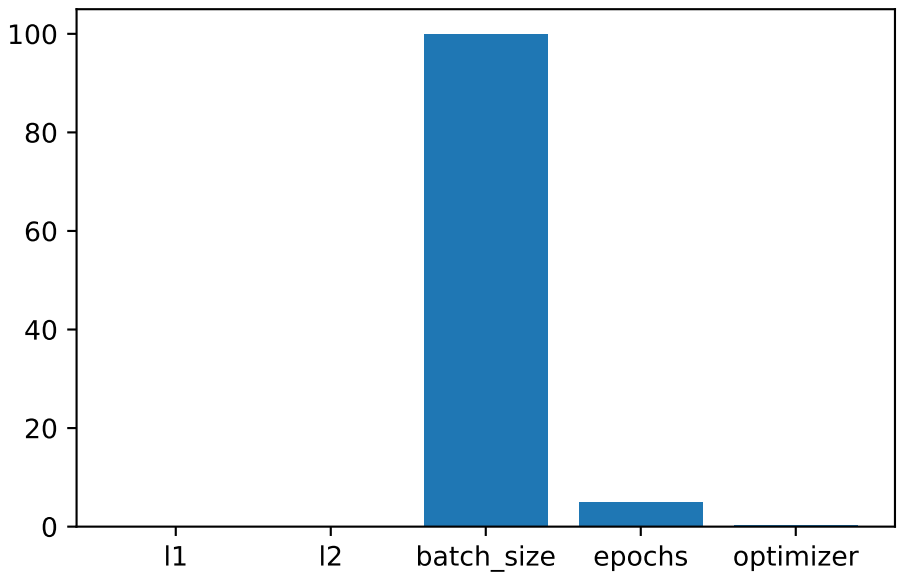



Figure 12.4: Variable importance plot, threshold 0.025.

### 12.10.1 Get the Tuned Architecture (SPOT Results)

The architecture of the `spotPython` model can be obtained as follows. First, the numerical representation of the hyperparameters are obtained, i.e., the numpy array `X` is generated. This array is then used to generate the model `model_spot` by the function `get_one_core_model_from_X`. The model `model_spot` has the following architecture:

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

```
Net_CIFAR10(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=8, bias=True)
  (fc3): Linear(in_features=8, out_features=10, bias=True)
)
```

### 12.10.2 Get Default Hyperparameters

In a similar manner as in Section 12.10.1, 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
from spotPython.hyperparameters.values import get_default_hyperparameters_as_array
X_start = get_default_hyperparameters_as_array(fun_control)
model_default = get_one_core_model_from_X(X_start, fun_control)
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)
)
```

### 12.10.3 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: 0.0975499972701073 | Loss: 2.3062030729293825 | Acc: 0.0975500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.0975499972701073 | Loss: 2.3040160190582277 | Acc: 0.0975500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.0987500026822090 | Loss: 2.3020986358642577 | Acc: 0.0987500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.1266500055789948 | Loss: 2.2995942102432250 | Acc: 0.1266500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.1498499959707260 | Loss: 2.2961405302047728 | Acc: 0.1498500000000000.
Epoch: 6 |

MulticlassAccuracy: 0.1425999999046326 | Loss: 2.2900444021224975 | Acc: 0.1426000000000000.
Epoch: 7 |

MulticlassAccuracy: 0.1688500046730042 | Loss: 2.2748941745758056 | Acc: 0.1688500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.1881999969482422 | Loss: 2.2260843358993530 | Acc: 0.1882000000000000.
Returned to Spot: Validation loss: 2.226084335899353

MulticlassAccuracy: 0.1918999999761581 | Loss: 2.2214019302368162 | Acc: 0.1919000000000000.
Final evaluation: Validation loss: 2.221401930236816
Final evaluation: Validation metric: 0.19189999997615814
----------------------------------------------

(2.221401930236816, nan, tensor(0.1919, device='mps:0'))
```

### 12.10.4 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: 0.3661000132560730 | Loss: 1.7093173021316528 | Acc: 0.3661000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.4627499878406525 | Loss: 1.4642572961807252 | Acc: 0.4627500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.4796499907970428 | Loss: 1.4531756073951720 | Acc: 0.4796500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5193499922752380 | Loss: 1.3521972542762757 | Acc: 0.5193500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5456500053405762 | Loss: 1.2886844976425171 | Acc: 0.5456500000000000.
Epoch: 6 |
```

```
MulticlassAccuracy: 0.5571500062942505 | Loss: 1.2521571839332581 | Acc: 0.5571500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5662500262260437 | Loss: 1.2315309381484985 | Acc: 0.5662500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5618000030517578 | Loss: 1.2532640023231507 | Acc: 0.5618000000000000.
Epoch: 9 |

MulticlassAccuracy: 0.5825999975204468 | Loss: 1.1913747765541076 | Acc: 0.5826000000000000.
Epoch: 10 |

MulticlassAccuracy: 0.5830000042915344 | Loss: 1.1833122503280640 | Acc: 0.5830000000000000.
Epoch: 11 |

MulticlassAccuracy: 0.5910000205039978 | Loss: 1.1831278825759888 | Acc: 0.5910000000000000.
Epoch: 12 |

MulticlassAccuracy: 0.5956000089645386 | Loss: 1.1784049792289735 | Acc: 0.5956000000000000.
Epoch: 13 |

MulticlassAccuracy: 0.5949500203132629 | Loss: 1.1580024556159974 | Acc: 0.5949500000000000.
Epoch: 14 |

MulticlassAccuracy: 0.5827500224113464 | Loss: 1.2288481868743897 | Acc: 0.5827500000000000.
Epoch: 15 |

MulticlassAccuracy: 0.5848000049591064 | Loss: 1.2282373707771301 | Acc: 0.5848000000000000.
Epoch: 16 |

MulticlassAccuracy: 0.5872499942779541 | Loss: 1.2612915629386903 | Acc: 0.5872500000000000.
Early stopping at epoch 15
Returned to Spot: Validation loss: 1.2612915629386903

MulticlassAccuracy: 0.5918999910354614 | Loss: 1.2511131338798962 | Acc: 0.5919000000000000.
Final evaluation: Validation loss: 1.2511131338798962
Final evaluation: Validation metric: 0.5918999910354614
----------------------------------------------

(1.2511131338798962, nan, tensor(0.5919, device='mps:0'))
```

### 12.10.5 Detailed Hyperparameter Plots

The contour plots in this section visualize the interactions of the three most important hyperparameters. Since some of these hyperparameters take fatorial or integer values, sometimes step-like fitness landcapes (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:  0.08257501318668711
l2:  0.08257501318668711
batch_size:  100.0
epochs:  5.036050457287037
optimizer:  0.2060782987482385
```

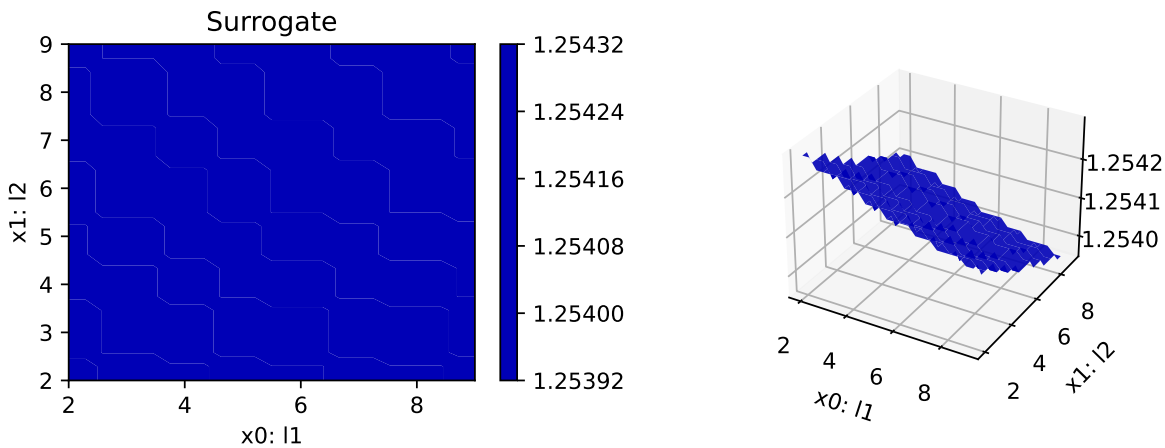
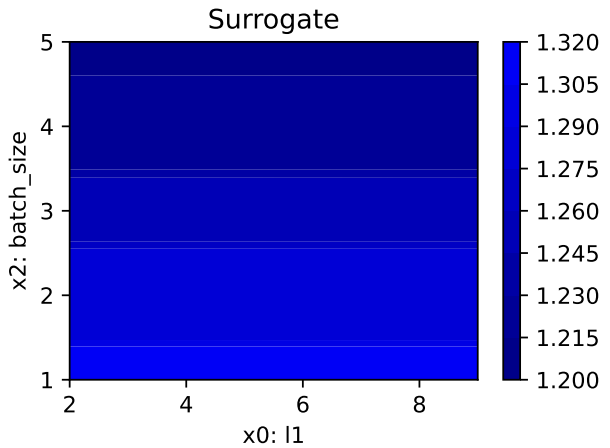
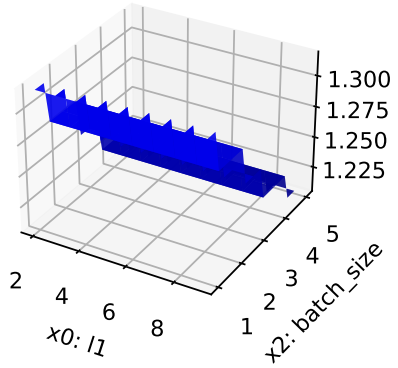
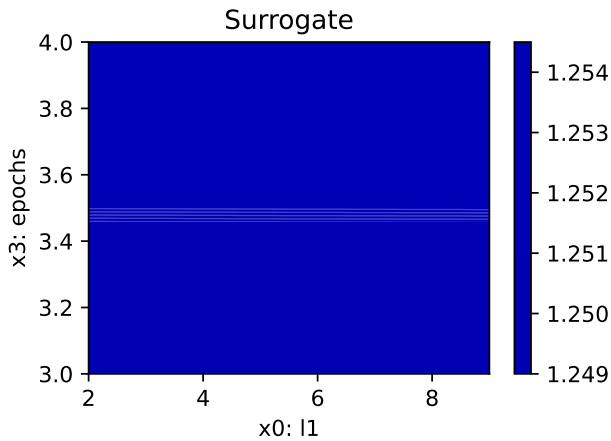
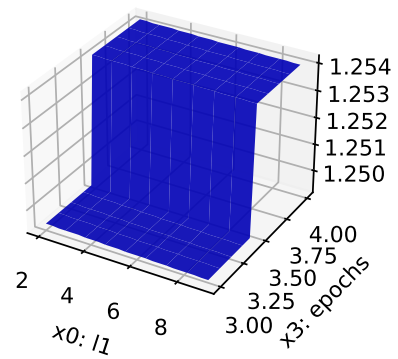


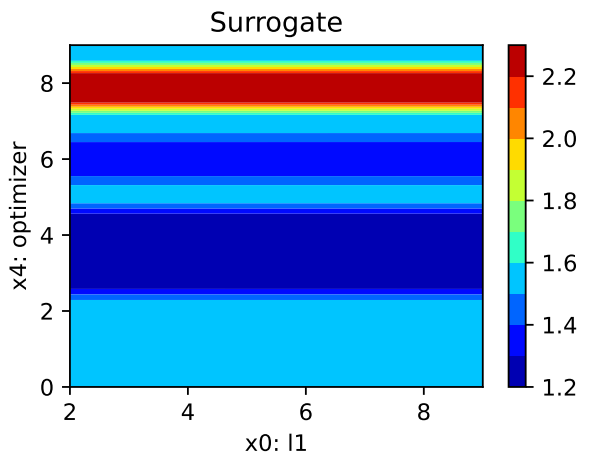

Figure 12.5: Contour plots.

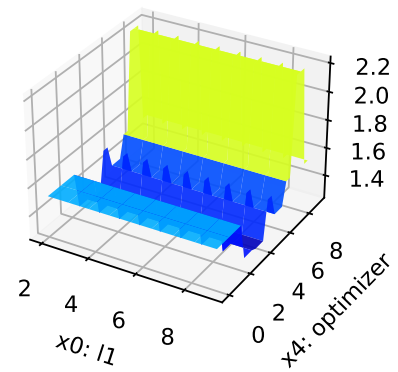

Surrogate
x2: batch_size
x0: l1
1.320
1.305
1.290
1.275
1.260
1.245
1.230
1.215
1.200

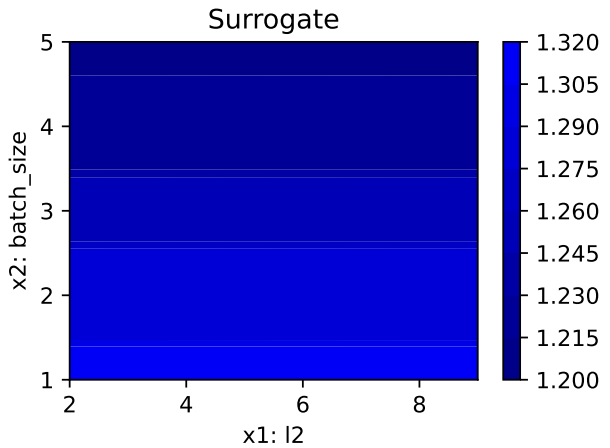

x0: l1
x2: batch_size
1.300
1.275
1.250
1.225

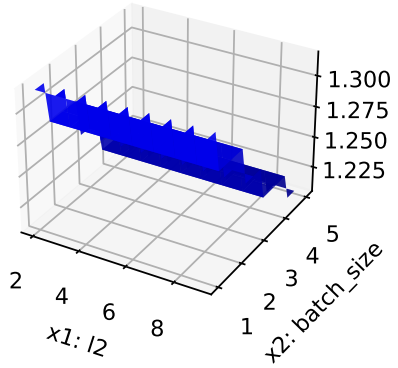

Surrogate
x3: epochs
x0: l1
1.254
1.253
1.252
1.251
1.250
1.249

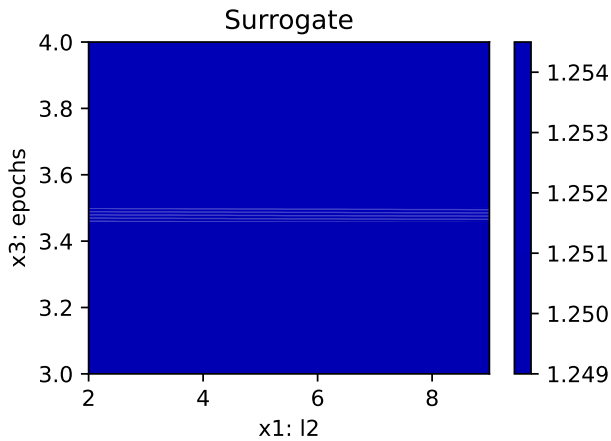

x0: l1
x3: epochs
1.254
1.253
1.252
1.251
1.250

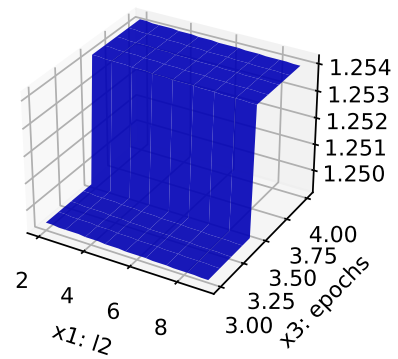

Surrogate
x4: optimizer
x0: l1
2.2
2.0
1.8
1.6
1.4
1.2

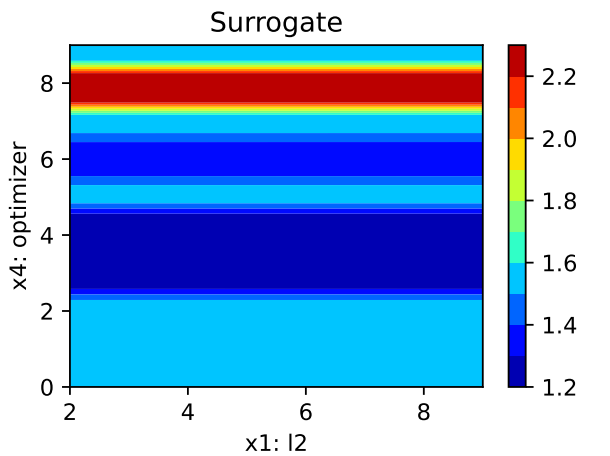

x0: l1
x4: optimizer
2.2
2.0
1.8
1.6
1.4

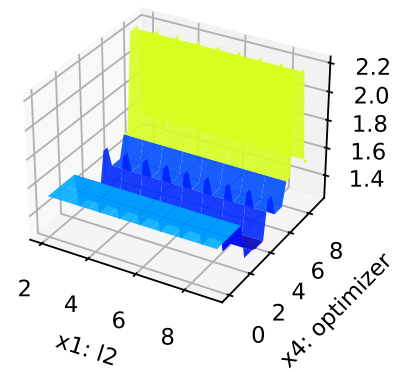

Surrogate
x2: batch_size
x1: l2
1.320
1.305
1.290
1.275
1.260
1.245
1.230
1.215
1.200

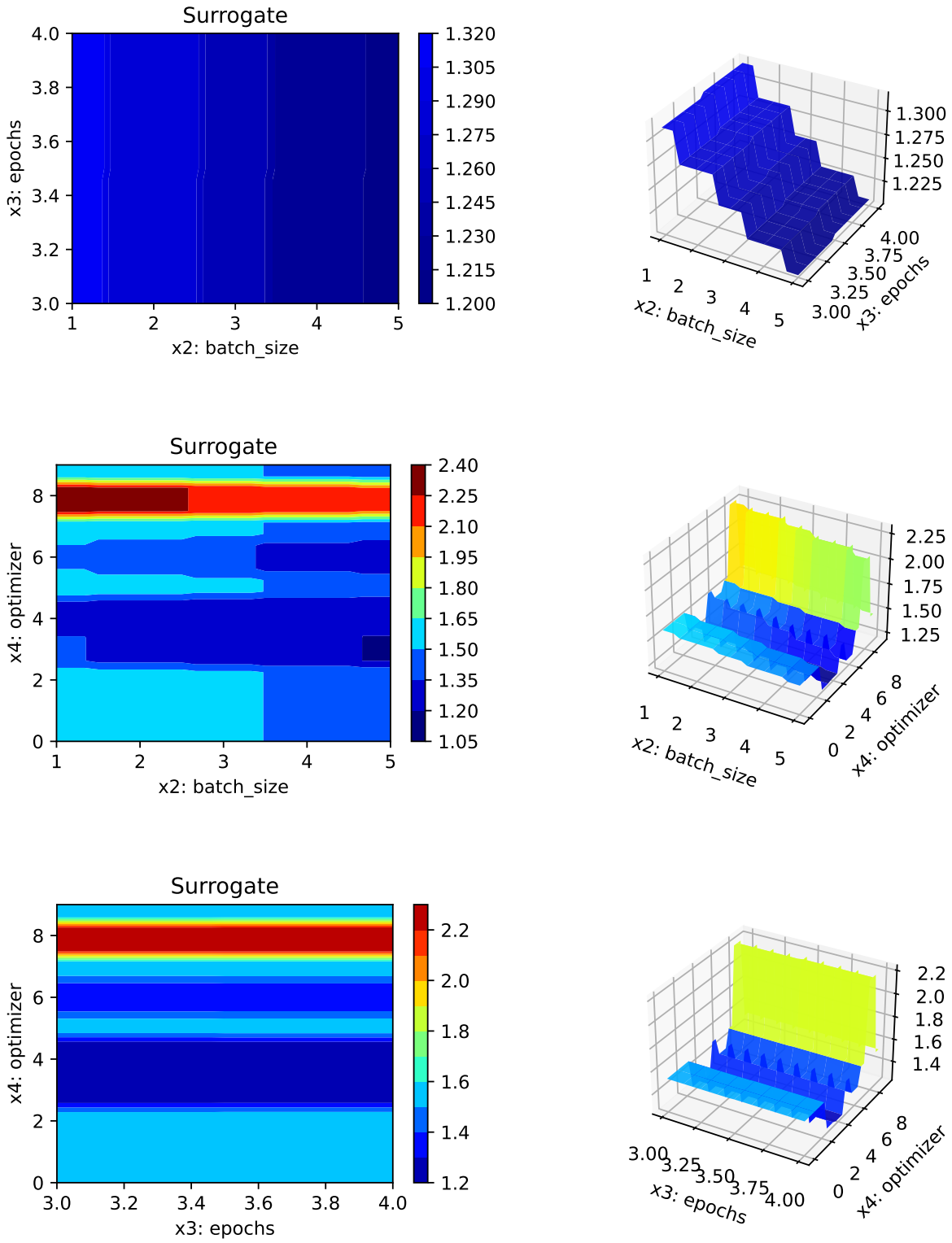

1.300
1.275
1.250
1.225
x1: l2
x2: batch_size


Surrogate
x3: epochs
x1: l2
1.254
1.253
1.252
1.251
1.250
1.249


1.254
1.253
1.252
1.251
1.250
x1: l2
x3: epochs


Surrogate
x4: optimizer
x1: l2
2.2
2.0
1.8
1.6
1.4
1.2


2.2
2.0
1.8
1.6
1.4
x1: l2
x4: optimizer


Surrogate
x3: epochs
x2: batch_size
4.0
3.8
3.6
3.4
3.2
3.0
1
2
3
4
5
1.320
1.305
1.290
1.275
1.260
1.245
1.230
1.215
1.200
1.300
1.275
1.250
1.225
4.00
3.75
3.50
3.25
3.00
Surrogate
x4: optimizer
x2: batch_size
8
6
4
2
0
2.40
2.25
2.10
1.95
1.80
1.65
1.50
1.35
1.20
1.05
2.25
2.00
1.75
1.50
1.25
Surrogate
x4: optimizer
x3: epochs
2.2
2.0
1.8
1.6
1.4
1.2

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
```

## 12.11 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.

## 12.12 Appendix

### 12.12.1 Sample Output From Ray Tune's Run

The output from `ray[tune]` could look like this (PyTorch 2023b):

```
Number of trials: 10 (10 TERMINATED)
------+------+-------------+--------------+---------+------------+--------------------+
|   l1 |   l2 |          lr |   batch_size |    loss |   accuracy | training_iteration |
+------+------+-------------+--------------+---------+------------+--------------------|
|   64 |    4 | 0.00011629  |            2 | 1.87273 |     0.244  |                  2 |
|   32 |   64 | 0.000339763 |            8 | 1.23603 |     0.567  |                  8 |
|    8 |   16 | 0.00276249  |           16 | 1.1815  |     0.5836 |                 10 |
|    4 |   64 | 0.000648721 |            4 | 1.31131 |     0.5224 |                  8 |
|   32 |   16 | 0.000340753 |            8 | 1.26454 |     0.5444 |                  8 |
|    8 |    4 | 0.000699775 |            8 | 1.99594 |     0.1983 |                  2 |
|  256 |    8 | 0.0839654   |           16 | 2.3119  |     0.0993 |                  1 |
|   16 |  128 | 0.0758154   |           16 | 2.33575 |     0.1327 |                  1 |
|   16 |    8 | 0.0763312   |           16 | 2.31129 |     0.1042 |                  4 |
|  128 |   16 | 0.000124903 |            4 | 2.26917 |     0.1945 |                  1 |
+-----+------+------+-------------+--------------+---------+------------+-----------------
Best trial config: {'l1': 8, 'l2': 16, 'lr': 0.00276249, 'batch_size': 16, 'data_dir': '..
Best trial final validation loss: 1.181501
Best trial final validation accuracy: 0.5836
Best trial test set accuracy: 0.5806
```

# 13 HPT: sklearn RandomForestClassifier VBDP Data

This document refers to the following software versions:

- `python`: 3.10.10

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

```
spotPython                  0.3.4
spotRiver                   0.0.93
```

```
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
```

## 13.1 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(IN
experiment_name = experiment_name.replace(':', '-')
print(experiment_name)
if not os.path.exists('./figures'):
    os.makedirs('./figures')
```

```
16-rf-sklearn_p040025_1min_5init_2023-07-06_02-04-47
```

```
import warnings
warnings.filterwarnings("ignore")
```

## 13.2 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")
```

## 13.3 Step 3: PyTorch Data Loading

### 13.3.1 Load Data: Classification VBDP

```
import pandas as pd
if ORIGINAL == True:
    train_df = pd.read_csv('./data/VBDP/trainn.csv')
    test_df = pd.read_csv('./data/VBDP/testt.csv')
else:
    train_df = pd.read_csv('./data/VBDP/train.csv')
    # remove the id column
    train_df = train_df.drop(columns=['id'])
```

```
from sklearn.preprocessing import OrdinalEncoder
n_samples = train_df.shape[0]
n_features = train_df.shape[1] - 1
target_column = "prognosis"
# Encoder our prognosis labels as integers for easier decoding later
enc = OrdinalEncoder()
train_df[target_column] = enc.fit_transform(train_df[[target_column]])
train_df.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
print(train_df.shape)
train_df.head()
```

```
(707, 65)
```

| | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | x10 | ... | x56 | x57 | x58 | x59 | x60 | x61 | x62 | x63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1.0 | 1.0 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.0 | 1.0 | 1.0 | ... | 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 | 1.0 | 0.0 | 1.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 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 | 0.0 | 1.0 | 1.0 |
| 3 | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | ... | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 | 0.0 |

The full data set `train_df` 64 features. The target column is labeled as `prognosis`.

### 13.3.2 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 | ... | x56 | x57 | x58 | x59 | x60 | x61 | x62 | x63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |
| 1 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 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 | 1.0 | 1.0 | 1.0 | 0.0 |
| 3 | 1.0 | 1.0 | 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 |
| 4 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | 1.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

```
# 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})
```

## 13.4 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 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(),
#     )
```

## 13.5 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:

`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.

```python
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
```

```python
# core_model  = RidgeCV
# core_model = GradientBoostingRegressor
# core_model = ElasticNet
core_model = RandomForestClassifier
# core_model = SVC
# core_model = LogisticRegression
# core_model = KNeighborsClassifier
# core_model = GradientBoostingClassifier
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:

```python
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
```

## 13.6 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model`

### 13.6.1 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:

`modify_hyper_parameter_bounds(fun_control, "tol", bounds=[1e-3, 1e-2])`

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

### 13.6.2 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 12.6.

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:

`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:
# modify_hyper_parameter_levels(fun_control, "loss", ["log_loss"])
```

> **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 13.7.3.

```
modify_hyper_parameter_bounds(fun_control, "bootstrap", bounds=[0, 1])
modify_hyper_parameter_bounds(fun_control, "oob_score", bounds=[0, 0])
```

### 13.6.3 Optimizers

Optimizers are described in Section 12.6.1.

### 13.6.4 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.

## 13.7 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"`.

### 13.7.1 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.

#### 13.7.1.1 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}`.

#### 13.7.1.2 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},
               })
```

### 13.7.2 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",
})
```

### 13.7.3 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.

> **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",
})
modify_hyper_parameter_bounds(fun_control, "bootstrap", bounds=[1, 1])
modify_hyper_parameter_bounds(fun_control, "oob_score", bounds=[1, 1])
```

#### 13.7.3.1 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,
# })
```

## 13.8 Step 8: Calling the SPOT Function

### 13.8.1 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)
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
```

### 13.8.2 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
```

### 13.8.3 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
X_start = get_default_hyperparameters_as_array(fun_control)
X_start
```

```
array([[ 7.,  0., 10.,  2.,  1.,  0.,  0., 10.,  0.,  1.,  0.]])
```

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

```
spotPython tuning: -0.34276729559748426 [----------] 1.40%

spotPython tuning: -0.34276729559748426 [----------] 1.89%

spotPython tuning: -0.34276729559748426 [----------] 2.47%

spotPython tuning: -0.34276729559748426 [----------] 3.00%

spotPython tuning: -0.34276729559748426 [----------] 3.40%
```

```
spotPython tuning: -0.35062893081761004 [----------] 4.07%

spotPython tuning: -0.35062893081761004 [----------] 4.85%

spotPython tuning: -0.35062893081761004 [#---------] 5.66%

spotPython tuning: -0.35062893081761004 [#---------] 6.98%

spotPython tuning: -0.35062893081761004 [#---------] 7.63%

spotPython tuning: -0.35062893081761004 [#---------] 8.92%

spotPython tuning: -0.35062893081761004 [#---------] 9.87%

spotPython tuning: -0.35062893081761004 [#---------] 12.79%

spotPython tuning: -0.35062893081761004 [##--------] 17.75%

spotPython tuning: -0.35062893081761004 [##--------] 20.06%

spotPython tuning: -0.35062893081761004 [##--------] 23.45%

spotPython tuning: -0.35062893081761004 [###-------] 26.98%

spotPython tuning: -0.35188679245283017 [###-------] 30.56%

spotPython tuning: -0.35188679245283017 [###-------] 34.81%

spotPython tuning: -0.35188679245283017 [####------] 40.00%

spotPython tuning: -0.35188679245283017 [####------] 44.16%

spotPython tuning: -0.35188679245283017 [#####-----] 48.61%

spotPython tuning: -0.35754716981132073 [#####-----] 52.76%

spotPython tuning: -0.35754716981132073 [######----] 56.81%
```

```
spotPython tuning: -0.35754716981132073 [######----] 61.07%

spotPython tuning: -0.35754716981132073 [#######---] 65.29%

spotPython tuning: -0.35754716981132073 [#######---] 70.08%

spotPython tuning: -0.35754716981132073 [########--] 76.78%

spotPython tuning: -0.35754716981132073 [########--] 83.29%

spotPython tuning: -0.35754716981132073 [#########-] 90.37%

spotPython tuning: -0.3591194968553459 [##########] 97.10%

spotPython tuning: -0.3591194968553459 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x29f34bb80>
```

## 13.9 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section 12.9, see also the description in the documentation: Tensorboard.

## 13.10 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")
```

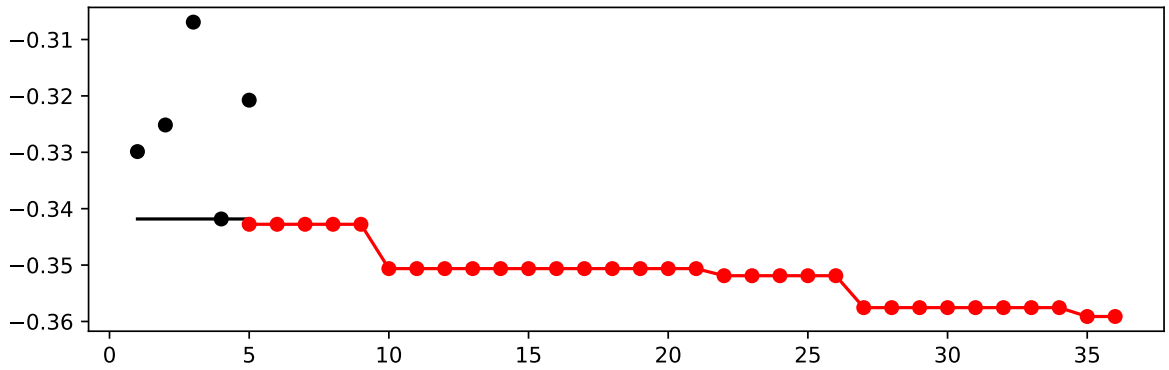



Figure 13.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- Print the results

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

```
| name                      | type   | default    |   lower |   upper |                  tuned
|---------------------------|--------|------------|---------|---------|-----------------------
| n_estimators              | int    | 7          |     5.0 |    10.0 |                    9.0
| criterion                 | factor | gini       |     0.0 |     2.0 |                    1.0
| max_depth                 | int    | 10         |     1.0 |    20.0 |                   12.0
| min_samples_split         | int    | 2          |     2.0 |   100.0 |                   11.0
| min_samples_leaf          | int    | 1          |     1.0 |    25.0 |                    1.0
| min_weight_fraction_leaf  | float  | 0.0        |     0.0 |    0.01 | 0.0059164181306543325
| max_features              | factor | sqrt       |     0.0 |     1.0 |                    0.0
| max_leaf_nodes            | int    | 10         |     7.0 |    12.0 |                    9.0
| min_impurity_decrease     | float  | 0.0        |     0.0 |    0.01 |                    0.0
| bootstrap                 | factor | 1          |     1.0 |     1.0 |                    1.0
| oob_score                 | factor | 0          |     1.0 |     1.0 |                    1.0
```

### 13.10.1 Show variable importance

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

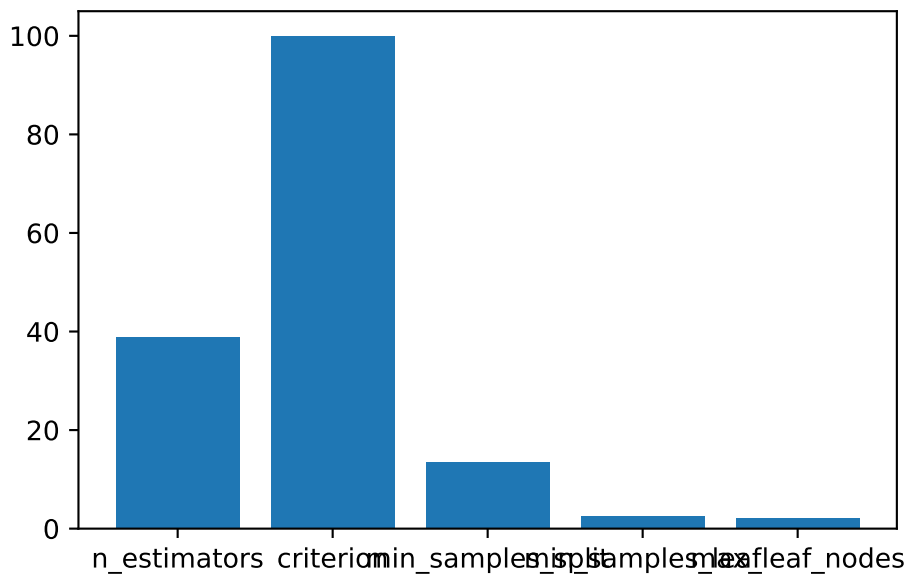



Figure 13.2: Variable importance plot, threshold 0.025.

### 13.10.2 Get Default Hyperparameters

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

```
{'n_estimators': 128,
 'criterion': 'gini',
 'max_depth': 1024,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'min_weight_fraction_leaf': 0.0,
 'max_features': 'sqrt',
 'max_leaf_nodes': 1024,
 'min_impurity_decrease': 0.0,
 'bootstrap': 1,
 'oob_score': 0}
```

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

```
Pipeline(steps=[('nonetype', None),
                ('randomforestclassifier',
                 RandomForestClassifier(bootstrap=1, max_depth=1024,
                                        max_leaf_nodes=1024, n_estimators=128,
                                        oob_score=0))])
```

### 13.10.3 Get SPOT Results

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

```
[[9.00000000e+00 1.00000000e+00 1.20000000e+01 1.10000000e+01
  1.00000000e+00 5.91641813e-03 0.00000000e+00 9.00000000e+00
  0.00000000e+00 1.00000000e+00 1.00000000e+00]]
```

```
from spotPython.hyperparameters.values import assign_values, return_conf_list_from_var_dic
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': 512,
  'criterion': 'entropy',
  'max_depth': 4096,
  'min_samples_split': 11,
  'min_samples_leaf': 1,
  'min_weight_fraction_leaf': 0.0059164181306543325,
  'max_features': 'sqrt',
  'max_leaf_nodes': 512,
  '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=4096,
                       max_leaf_nodes=512, min_samples_split=11,
                       min_weight_fraction_leaf=0.0059164181306543325,
                       n_estimators=512, oob_score=1)
```

### 13.10.4 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
```

```
0.3615819209039548
```

```
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
```

### 13.10.5 Handling Non-deterministic Results

- Because the model is non-determinstic, we perform $n = 30$ runs and calculate the mean and standard deviation of the performance metric.

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

```
mean_res: 0.3583804143126177
std_res: 0.007006898349060751
min_res: 0.34463276836158196
max_res: 0.3757062146892655
median_res: 0.3578154425612053
```

### 13.10.6 Evalution 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)
```

```
0.3436911487758945
```

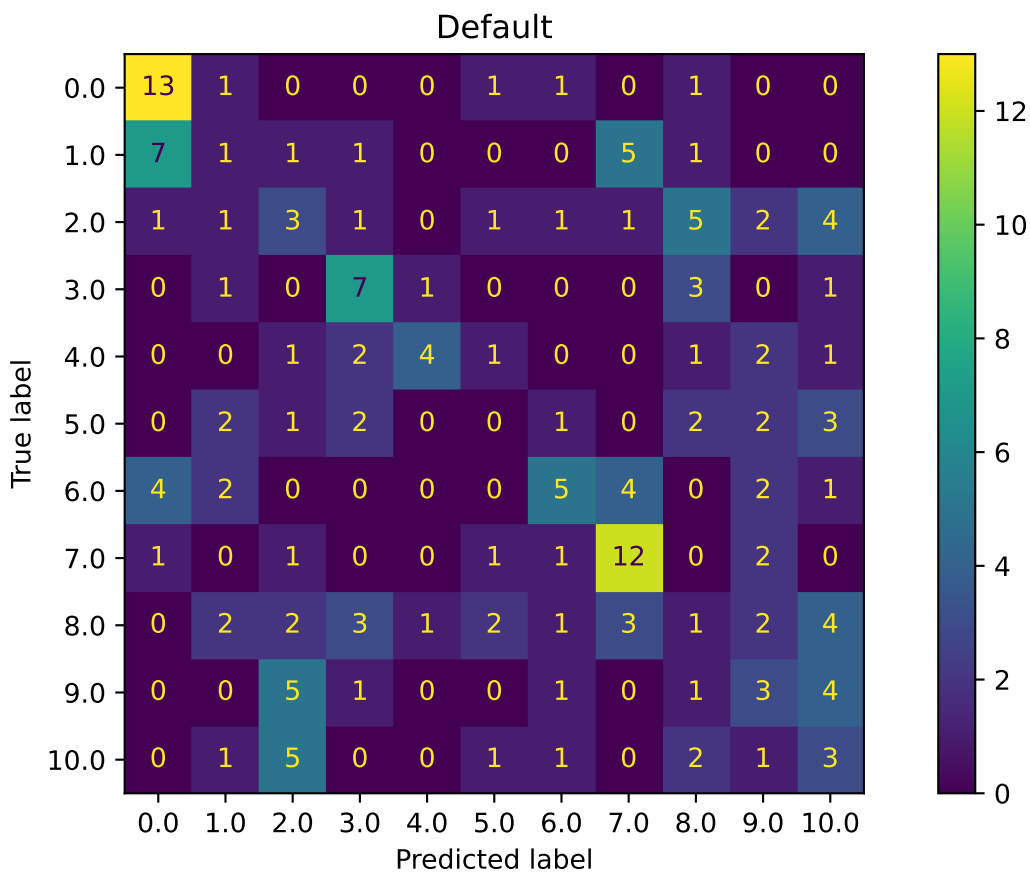
Since one single evaluation is not meaningful, we perform, similar to the evaluation of the SPOT results, $n = 30$ runs of the default setting and and calculate the mean and standard deviation of the performance metric.

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

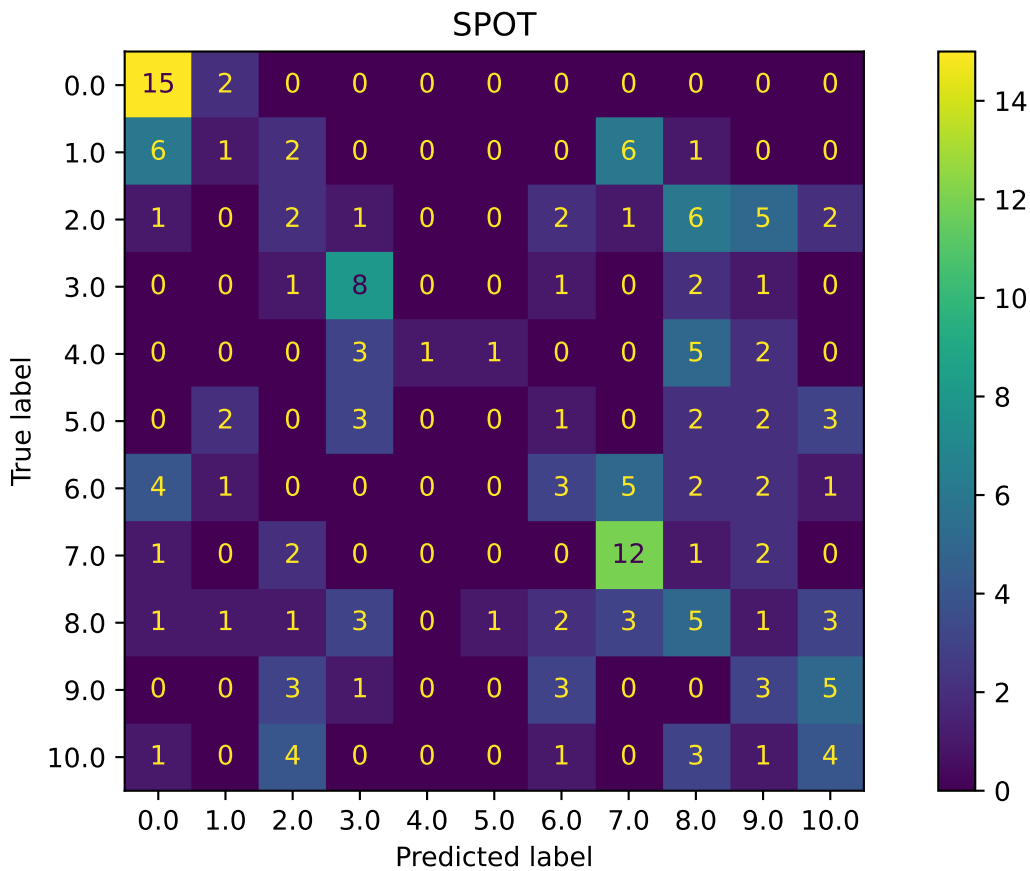
```
mean_res: 0.34497802887633394
std_res: 0.01267398819635564
min_res: 0.3229755178907721
max_res: 0.36817325800376643
median_res: 0.3427495291902071
```

### 13.10.7 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)
```

```
(-0.3591194968553459, -0.28962264150943395)
```

### 13.10.8 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)
```

```
(0.359433962264151, None)
```

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

```
(0.31290849673202614, None)
```

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

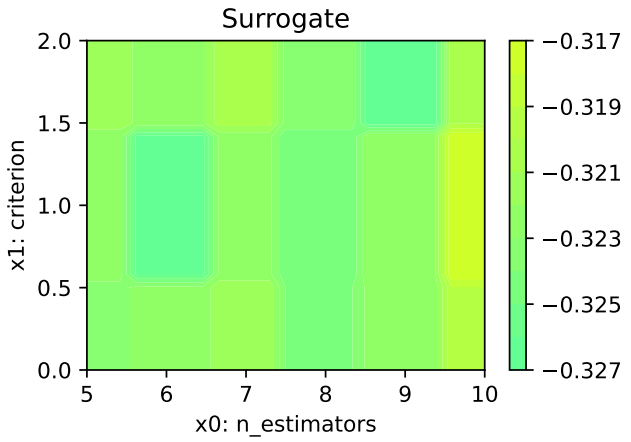
```
fun_control.update({
     "eval": "data_cv",
     "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)
```

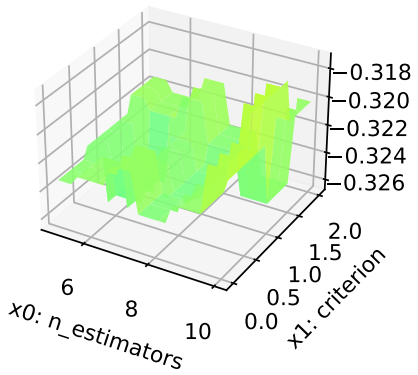
```
(0.3648658618376928, None)
```

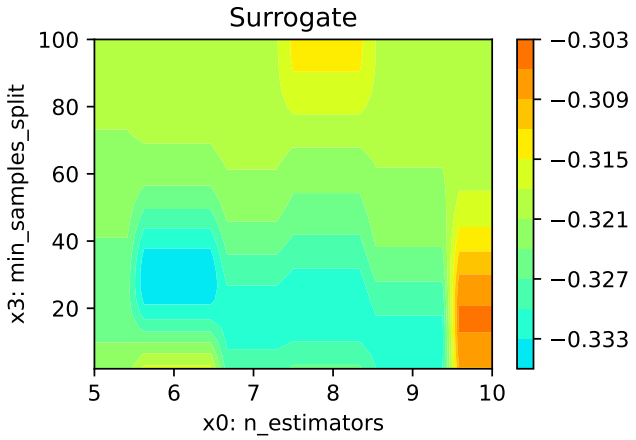
### 13.10.9 Detailed Hyperparameter Plots

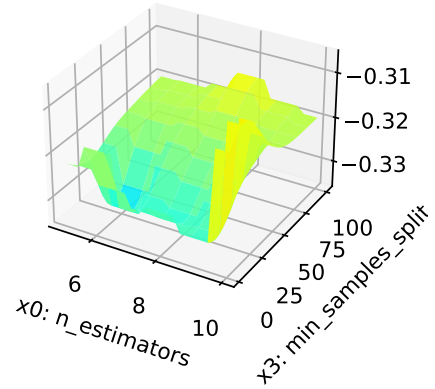
```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

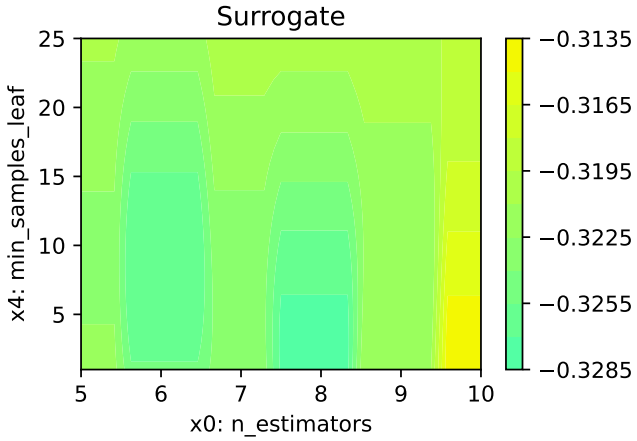
```
n_estimators:  38.85223485225775
criterion:  100.0
min_samples_split:  13.424756161824943
min_samples_leaf:  2.5425186088124936
max_leaf_nodes:  2.1411547457633917
```

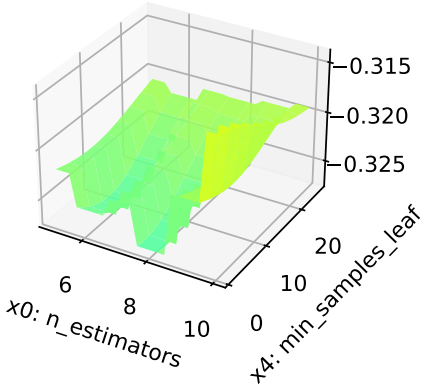

Surrogate
x1: criterion
x0: n_estimators
2.0
1.5
1.0
0.5
0.0
5
6
7
8
9
10
−0.317
−0.319
−0.321
−0.323
−0.325
−0.327

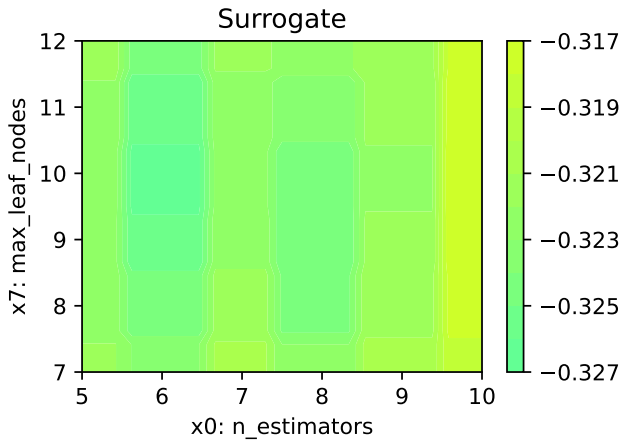

−0.318
−0.320
−0.322
−0.324
−0.326
x0: n_estimators
x1: criterion
6
8
10
2.0
1.5
1.0
0.5
0.0

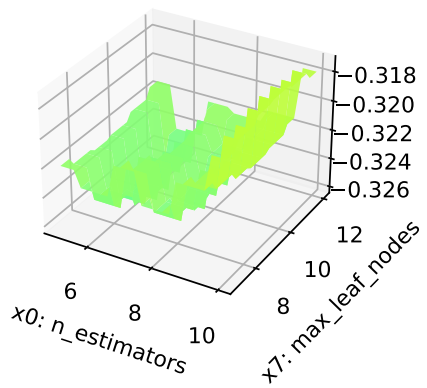

Surrogate
x3: min_samples_split
x0: n_estimators
100
80
60
40
20
5
6
7
8
9
10
−0.303
−0.309
−0.315
−0.321
−0.327
−0.333

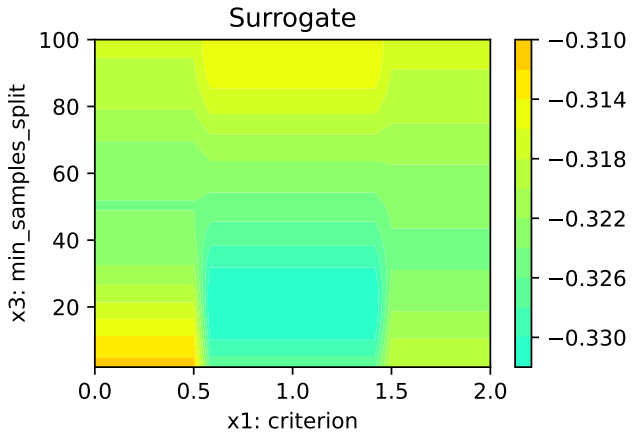

−0.31
−0.32
−0.33
x0: n_estimators
x3: min_samples_split
6
8
10
100
75
50
25
0

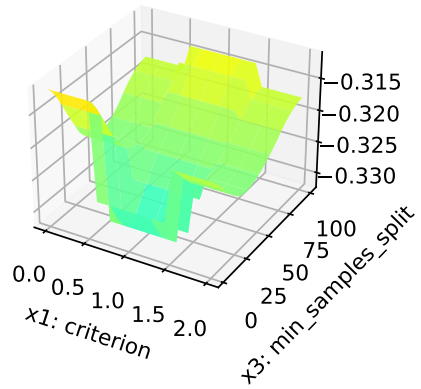

Surrogate
x4: min_samples_leaf
x0: n_estimators
25
20
15
10
5
5
6
7
8
9
10
−0.3135
−0.3165
−0.3195
−0.3225
−0.3255
−0.3285

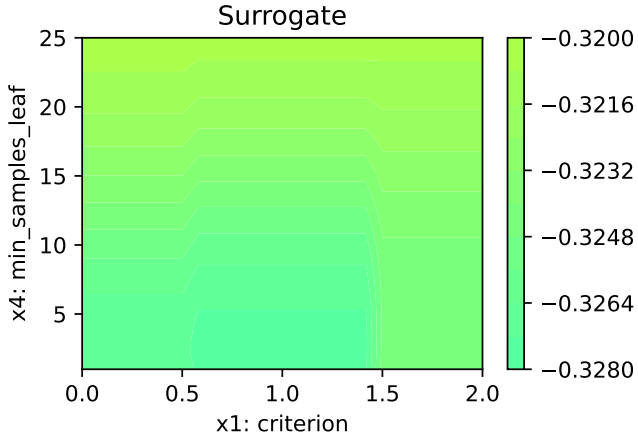

−0.315
−0.320
−0.325
x0: n_estimators
x4: min_samples_leaf
6
8
10
20
10
0

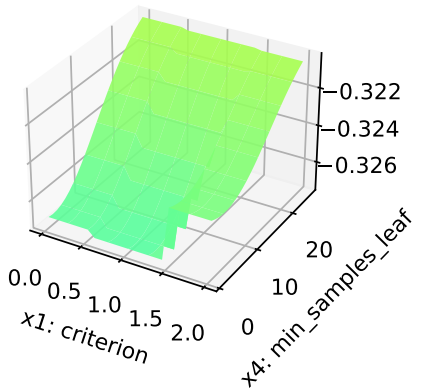

Surrogate
x7: max_leaf_nodes
x0: n_estimators
12
11
10
9
8
7
5
6
7
8
9
10
−0.317
−0.319
−0.321
−0.323
−0.325
−0.327

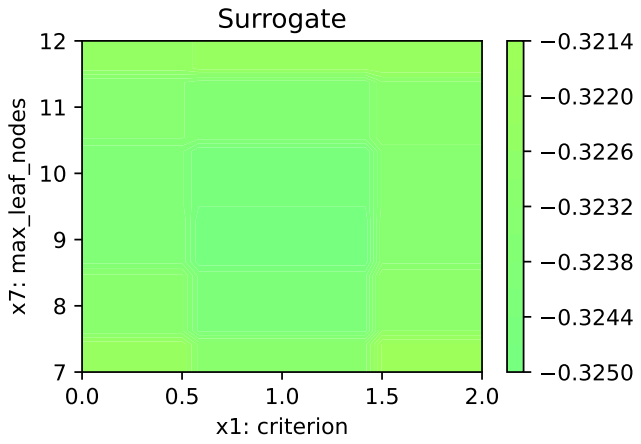

−0.318
−0.320
−0.322
−0.324
−0.326
6
8
10
x0: n_estimators
8
10
12
x7: max_leaf_nodes

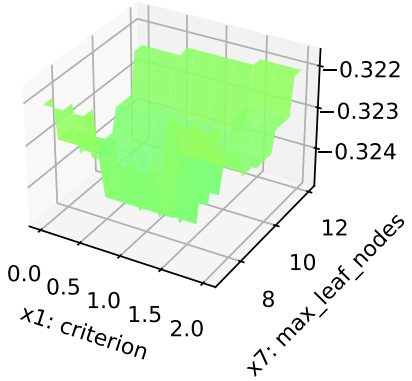

Surrogate
x3: min_samples_split
x1: criterion
100
80
60
40
20
0.0
0.5
1.0
1.5
2.0
−0.310
−0.314
−0.318
−0.322
−0.326
−0.330

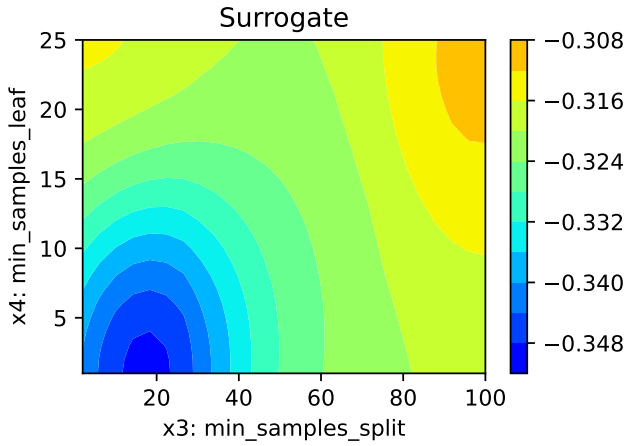

−0.315
−0.320
−0.325
−0.330
0.0
0.5
1.0
1.5
2.0
x1: criterion
0
25
50
75
100
x3: min_samples_split

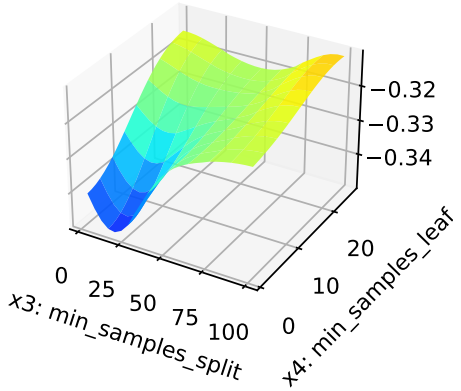

Surrogate
x4: min_samples_leaf
x1: criterion
25
20
15
10
5
0.0
0.5
1.0
1.5
2.0
−0.3200
−0.3216
−0.3232
−0.3248
−0.3264
−0.3280

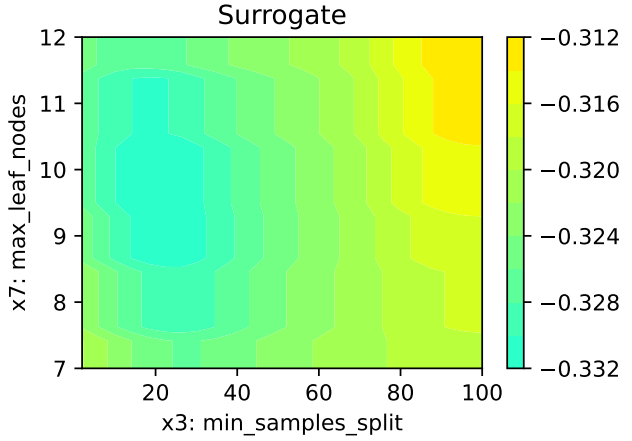

−0.322
−0.324
−0.326
0.0
0.5
1.0
1.5
2.0
x1: criterion
0
10
20
x4: min_samples_leaf

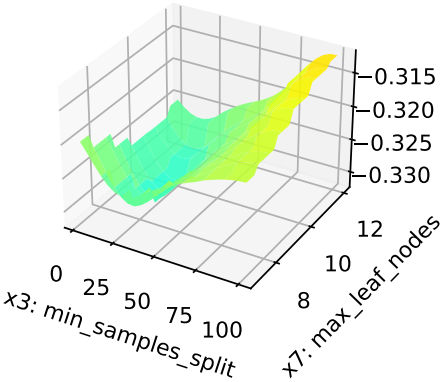

Surrogate
x7: max_leaf_nodes
x1: criterion
12
11
10
9
8
7
0.0
0.5
1.0
1.5
2.0
−0.3214
−0.3220
−0.3226
−0.3232
−0.3238
−0.3244
−0.3250

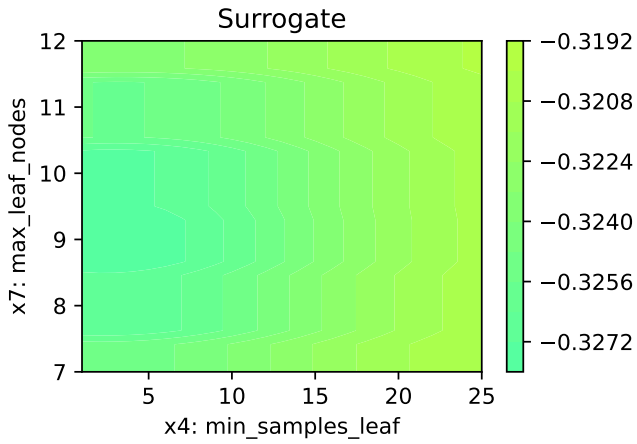

−0.322
−0.323
−0.324
0.0
0.5
1.0
1.5
2.0
x1: criterion
8
10
12
x7: max_leaf_nodes

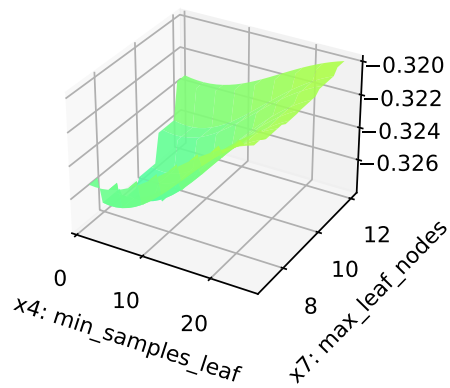

Surrogate
x4: min_samples_leaf
x3: min_samples_split
25
20
15
10
5
20
40
60
80
100
−0.308
−0.316
−0.324
−0.332
−0.340
−0.348


−0.32
−0.33
−0.34
0
25
50
75
100
x3: min_samples_split
0
10
20
x4: min_samples_leaf


Surrogate
x7: max_leaf_nodes
x3: min_samples_split
12
11
10
9
8
7
20
40
60
80
100
−0.312
−0.316
−0.320
−0.324
−0.328
−0.332


−0.315
−0.320
−0.325
−0.330
0
25
50
75
100
x3: min_samples_split
8
10
12
x7: max_leaf_nodes




### 13.10.10 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
```

### 13.10.11 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)
```

# 14 HPT: sklearn XGB Classifier VBDP Data

This document refers to the following software versions:

- python: 3.10.10

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

```
spotPython                  0.3.4
spotRiver                   0.0.93

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
```

## 14.1 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_p040025_1min_5init_2023-07-06_02-10-28
```

```
import warnings
warnings.filterwarnings("ignore")
```

## 14.2 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")
```

## 14.3 Step 3: PyTorch Data Loading

### 14.3.1 1. Load Data: Classification VBDP

```
import pandas as pd
if ORIGINAL == True:
    train_df = pd.read_csv('./data/VBDP/trainn.csv')
    test_df = pd.read_csv('./data/VBDP/testt.csv')
else:
    train_df = pd.read_csv('./data/VBDP/train.csv')
    # remove the id column
    train_df = train_df.drop(columns=['id'])
```

```
from sklearn.preprocessing import OrdinalEncoder
n_samples = train_df.shape[0]
n_features = train_df.shape[1] - 1
target_column = "prognosis"
# Encoder our prognosis labels as integers for easier decoding later
enc = OrdinalEncoder()
train_df[target_column] = enc.fit_transform(train_df[[target_column]])
train_df.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
print(train_df.shape)
train_df.head()
```

```
(707, 65)
```

| | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | x10 | ... | x56 | x57 | x58 | x59 | x60 | x61 | x62 | x63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1.0 | 1.0 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.0 | 1.0 | 1.0 | ... | 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 | 1.0 | 0.0 | 1.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 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 | 0.0 | 1.0 | 1.0 |
| 3 | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | ... | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 | 0.0 |

The full data set `train_df` 64 features. The target column is labeled as `prognosis`.

### 14.3.2 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 | ... | x56 | x57 | x58 | x59 | x60 | x61 | x62 | x63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |
| 1 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 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 | 1.0 | 1.0 | 1.0 | 0.0 |
| 3 | 1.0 | 1.0 | 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 |
| 4 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | 1.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

```
# 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})
```

## 14.4 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 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(),
#     )
```

## 14.5 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:

`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.

```python
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
```

```python
# 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
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:

```python
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
```

## 14.6 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model`

### 14.6.1 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:

```
modify_hyper_parameter_bounds(fun_control, "tol", bounds=[1e-3, 1e-2])
```

```
from spotPython.hyperparameters.values import modify_hyper_parameter_bounds
# modify_hyper_parameter_bounds(fun_control, "tol", bounds=[1e-3, 1e-2])
# modify_hyper_parameter_bounds(fun_control, "min_samples_split", bounds=[3, 20])
# modify_hyper_parameter_bounds(fun_control, "dual", bounds=[0, 0])
# modify_hyper_parameter_bounds(fun_control, "probability", bounds=[1, 1])
# fun_control["core_model_hyper_dict"]["tol"]
# modify_hyper_parameter_bounds(fun_control, "min_samples_leaf", bounds=[1, 25])
# modify_hyper_parameter_bounds(fun_control, "n_estimators", bounds=[5, 10])
```

### 14.6.2 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 12.6.

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:

```
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:
modify_hyper_parameter_levels(fun_control, "loss", ["log_loss"])
```

### 14.6.3 Optimizers

Optimizers are described in Section 12.6.1.

## 14.7 Step 7: Selection of the Objective (Loss) Function

### 14.7.1 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).

### 14.7.2 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.

### 14.7.3 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"`.

### 14.7.4 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`.

> **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.

#### 14.7.4.1 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}`.

#### 14.7.4.2 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.

> **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},
               })
```

### 14.7.5 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",
})
```

#### 14.7.5.1 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,
# })
```

## 14.8 Step 8: Calling the SPOT Function

### 14.8.1 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)
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                  |
```

### 14.8.2 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
```

### 14.8.3 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
X_start = get_default_hyperparameters_as_array(fun_control)
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: -0.40100250626566414 [#---------] 6.32%

spotPython tuning: -0.40100250626566414 [#---------] 8.48%

spotPython tuning: -0.40100250626566414 [#---------] 10.33%

spotPython tuning: -0.40100250626566414 [#---------] 13.39%

spotPython tuning: -0.40100250626566414 [#---------] 14.99%

spotPython tuning: -0.40100250626566414 [##--------] 20.14%

spotPython tuning: -0.40100250626566414 [##--------] 21.22%

spotPython tuning: -0.40100250626566414 [###-------] 26.68%

spotPython tuning: -0.40100250626566414 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x29634f700>
```

## 14.9 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section 12.9, see also the description in the documentation: Tensorboard.

## 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")
```

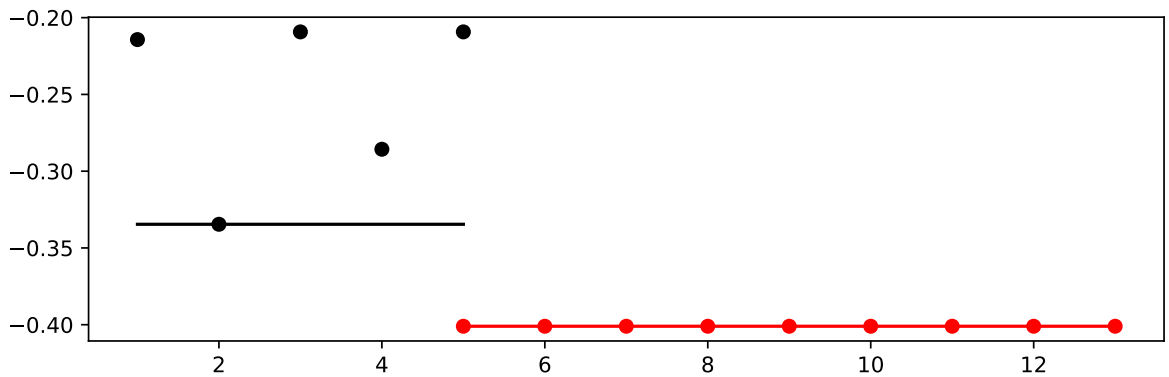



Figure 14.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- 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 |   -0.9302847173981572 | trans
| max_iter          | int    | 7          |     3.0 |    10.0 |                   9.0 | trans
| max_leaf_nodes    | int    | 5          |     1.0 |    12.0 |                   5.0 | trans
| max_depth         | int    | 2          |     1.0 |    20.0 |                  19.0 | trans
```

```
| min_samples_leaf  | int    | 4         |     2.0 |    10.0 |                   2.0 | transf
| l2_regularization | float  | 0.0       |     0.0 |    10.0 |    2.4029083174160553 | None
| max_bins          | int    | 255       |   127.0 |   255.0 |                 142.0 | None
| early_stopping    | factor | 1         |     0.0 |     1.0 |                   1.0 | None
| n_iter_no_change  | int    | 10        |     5.0 |    20.0 |                   6.0 | None
| tol               | float  | 0.0001    |   1e-05 |   0.001 | 0.0009512860974290124 | None
```

### 14.10.1 Show variable importance

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

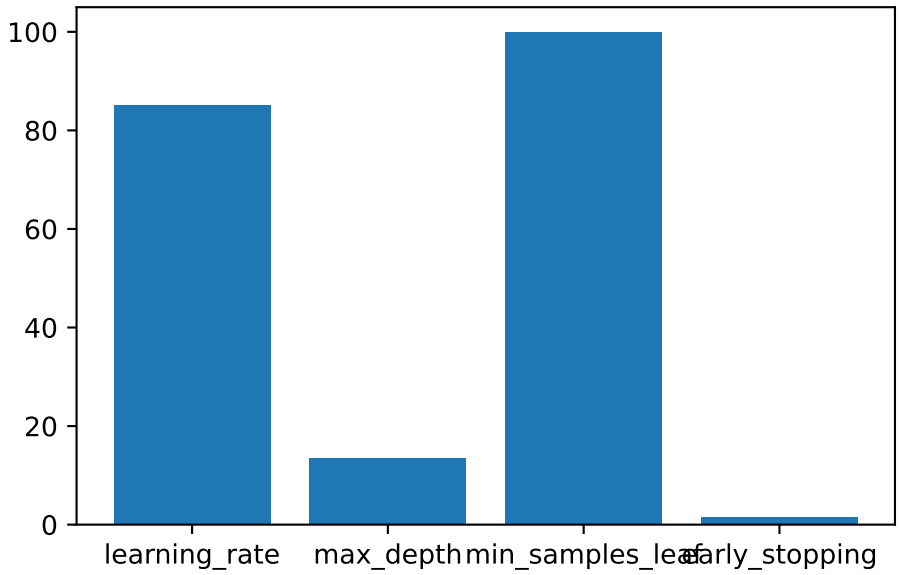



Figure 14.2: Variable importance plot, threshold 0.025.

### 14.10.2 Get Default Hyperparameters

```
from spotPython.hyperparameters.values import get_default_values, transform_hyper_paramete
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))])
```

### 14.10.3 Get SPOT Results

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

```
[[ 0.00000000e+00 -9.30284717e-01  9.00000000e+00  5.00000000e+00
   1.90000000e+01  2.00000000e+00  2.40290832e+00  1.42000000e+02
   1.00000000e+00  6.00000000e+00  9.51286097e-04]]
```

```
from spotPython.hyperparameters.values import assign_values, return_conf_list_from_var_dic
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': 0.11741275609341804,
  'max_iter': 512,
```

```
  'max_leaf_nodes': 32,
  'max_depth': 524288,
  'min_samples_leaf': 4,
  'l2_regularization': 2.4029083174160553,
  'max_bins': 142,
  'early_stopping': 1,
  'n_iter_no_change': 6,
  'tol': 0.0009512860974290124}]
```

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

```
HistGradientBoostingClassifier(early_stopping=1,
                               l2_regularization=2.4029083174160553,
                               learning_rate=0.11741275609341804, max_bins=142,
                               max_depth=524288, max_iter=512,
                               max_leaf_nodes=32, min_samples_leaf=4,
                               n_iter_no_change=6, tol=0.0009512860974290124)
```

### 14.10.4 Evaluate SPOT Results

- Fetch the data.

```python
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:

```python
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
```

```
0.36252354048964214
```

```python
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
```

### 14.10.5 Handling Non-deterministic Results

- Because the model is non-determinstic, we perform $n = 30$ runs and calculate the mean and standard deviation of the performance metric.

```python
_ = repeated_eval(30, model_spot)
```

```
mean_res: 0.33983050847457624
std_res: 0.013867406540579606
min_res: 0.3088512241054614
max_res: 0.36817325800376643
median_res: 0.3385122410546139
```

### 14.10.6 Evalution of the Default Hyperparameters

```python
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)
```

```
0.33427495291902076
```

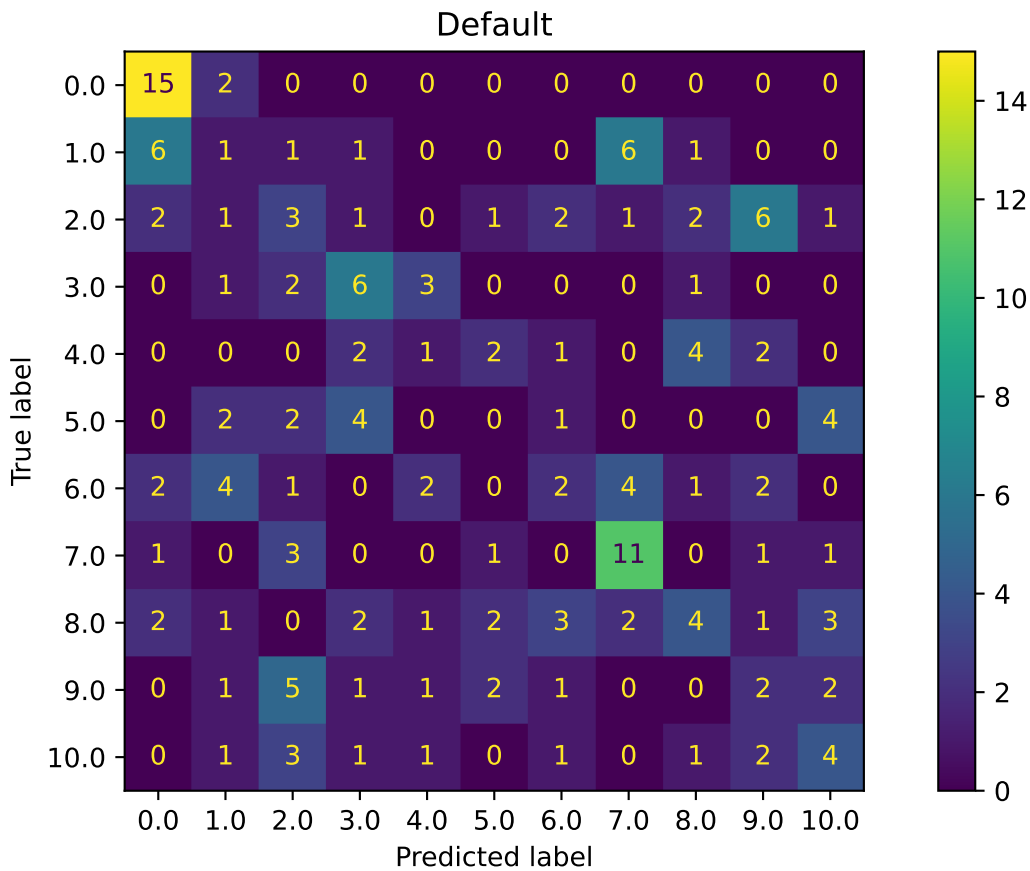
Since one single evaluation is not meaningful, we perform, similar to the evaluation of the SPOT results, $n = 30$ runs of the default setting and and calculate the mean and standard deviation of the performance metric.

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

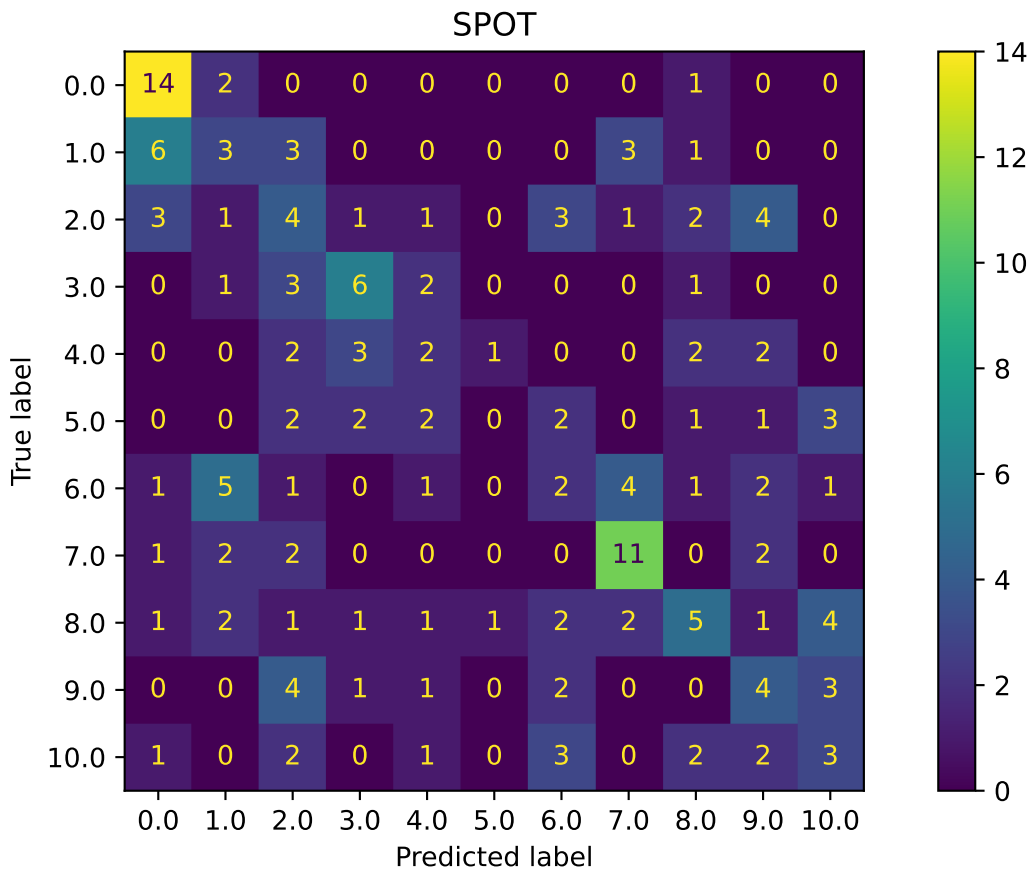
```
mean_res: 0.34588826114249843
std_res: 0.01672858740749801
min_res: 0.3163841807909605
max_res: 0.38229755178907715
median_res: 0.346045197740113
```

### 14.10.7 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)
```

```
(-0.40100250626566414, -0.20927318295739344)
```

### 14.10.8 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)
```

```
(0.3320754716981132, None)
```

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

```
(0.27777777777777773, None)
```

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

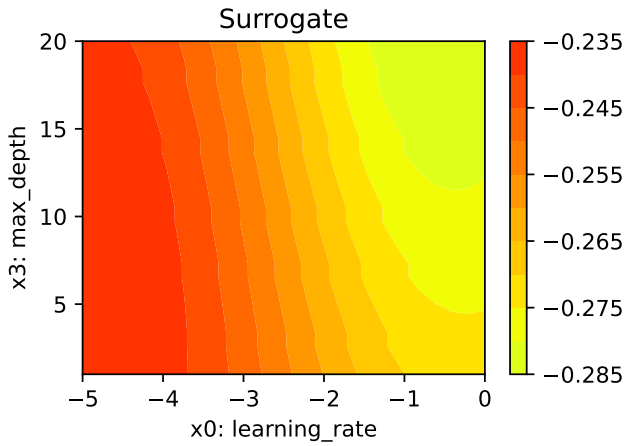
```
fun_control.update({
     "eval": "data_cv",
     "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)
```

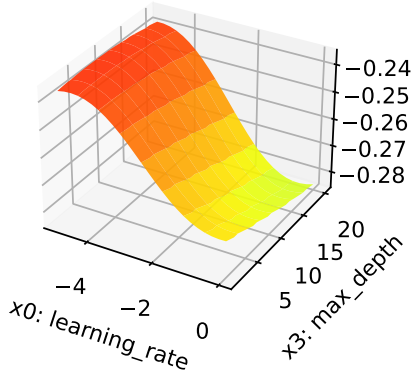
```
(0.3434473507712944, None)
```

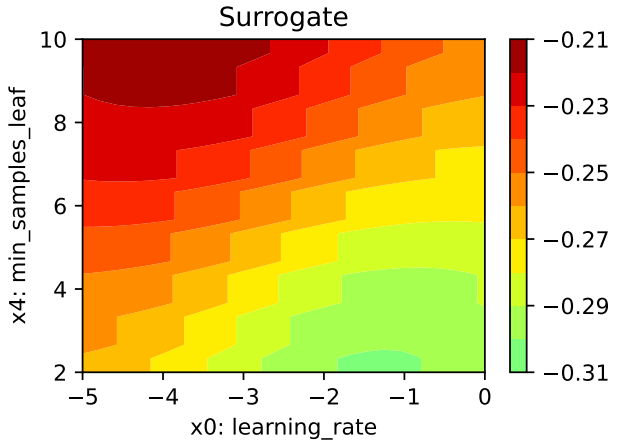
### 14.10.9 Detailed Hyperparameter Plots

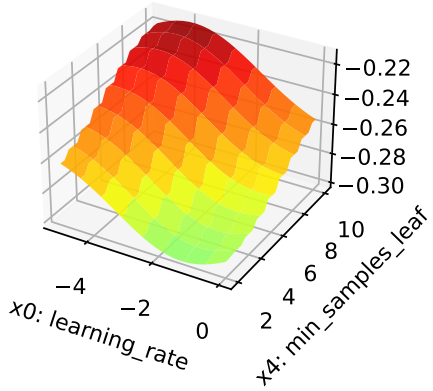
```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

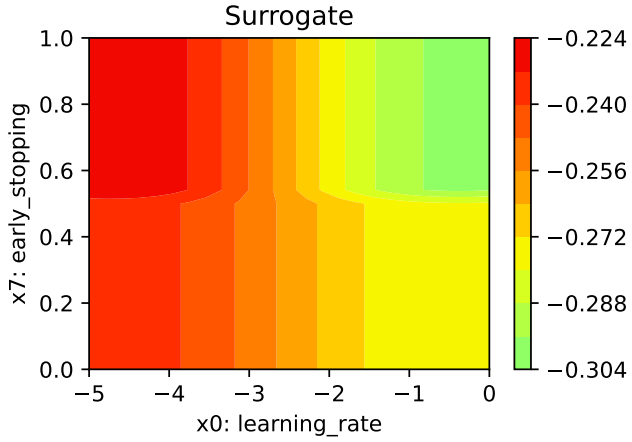
```
learning_rate:  85.19160867263129
max_depth:  13.509419214003497
min_samples_leaf:  100.0
early_stopping:  1.5184335190083467
```

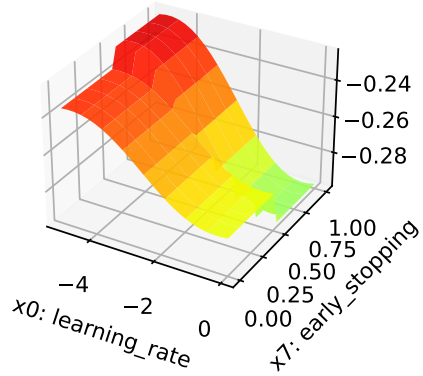

Surrogate
x3: max_depth
x0: learning_rate
20
15
10
5
−5
−4
−3
−2
−1
0
−0.235
−0.245
−0.255
−0.265
−0.275
−0.285

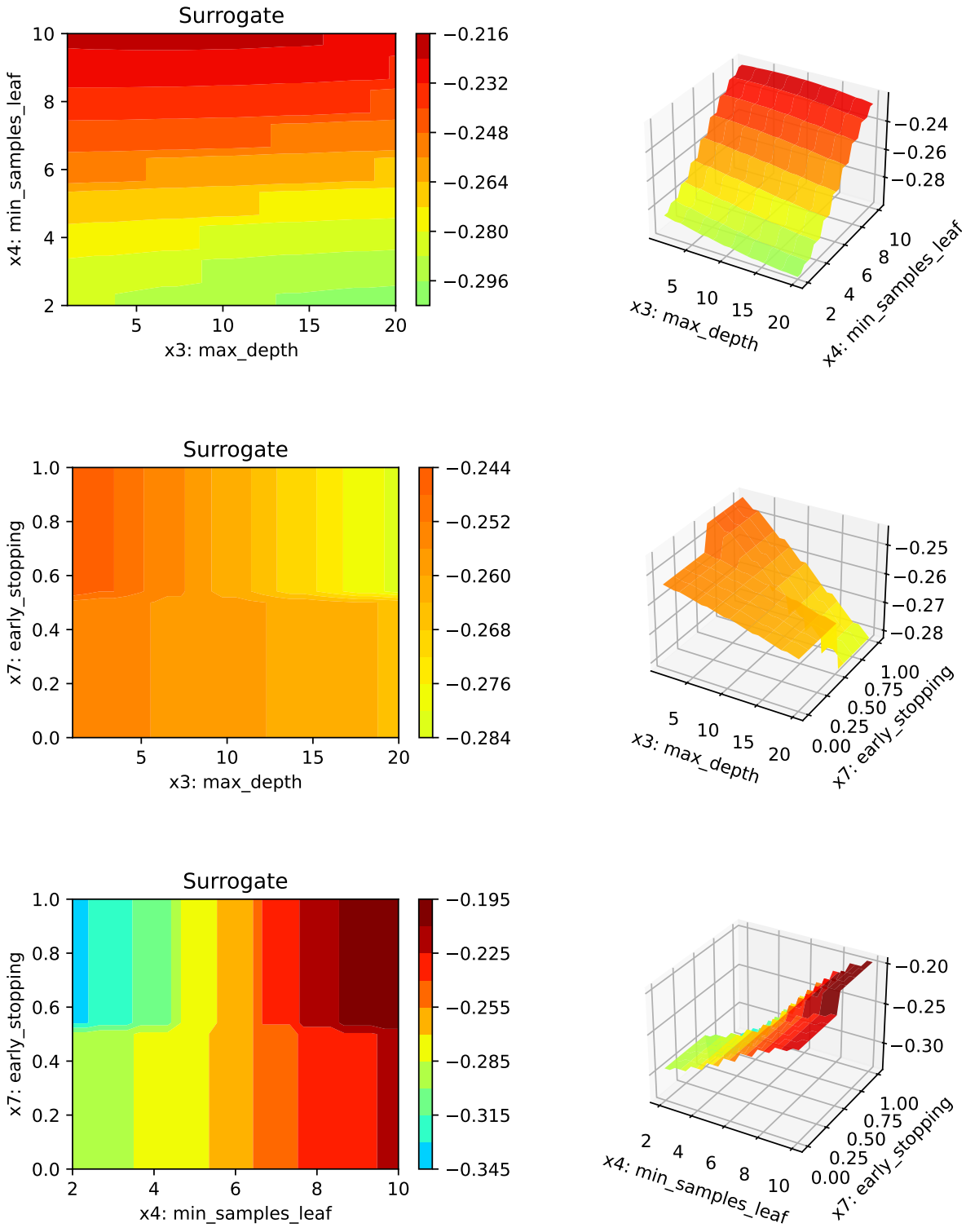

x0: learning_rate
x3: max_depth
−4
−2
0
20
15
10
5
−0.24
−0.25
−0.26
−0.27
−0.28


Surrogate
x4: min_samples_leaf
x0: learning_rate
10
8
6
4
2
−5
−4
−3
−2
−1
0
−0.21
−0.23
−0.25
−0.27
−0.29
−0.31


x0: learning_rate
x4: min_samples_leaf
−4
−2
0
10
8
6
4
2
−0.22
−0.24
−0.26
−0.28
−0.30


Surrogate
x7: early_stopping
x0: learning_rate
1.0
0.8
0.6
0.4
0.2
0.0
−5
−4
−3
−2
−1
0
−0.224
−0.240
−0.256
−0.272
−0.288
−0.304


x0: learning_rate
x7: early_stopping
−4
−2
0
1.00
0.75
0.50
0.25
0.00
−0.24
−0.26
−0.28


Surrogate
x4: min_samples_leaf
x3: max_depth
−0.216
−0.232
−0.248
−0.264
−0.280
−0.296
x3: max_depth
x4: min_samples_leaf
−0.24
−0.26
−0.28
Surrogate
x7: early_stopping
x3: max_depth
−0.244
−0.252
−0.260
−0.268
−0.276
−0.284
x3: max_depth
x7: early_stopping
−0.25
−0.26
−0.27
−0.28
Surrogate
x7: early_stopping
x4: min_samples_leaf
−0.195
−0.225
−0.255
−0.285
−0.315
−0.345
x4: min_samples_leaf
x7: early_stopping
−0.20
−0.25
−0.30

### 14.10.10 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
```

### 14.10.11 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)
```

# 15 HPT: sklearn SVC VBDP Data

This document refers to the following software versions:

- python: 3.10.10

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

```
spotPython                  0.3.4
spotRiver                   0.0.93

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 = '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_p040025_1min_5init_2023-07-06_02-15-16
```

```
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 1. Load Data: Classification VBDP

```
import pandas as pd
if ORIGINAL == True:
    train_df = pd.read_csv('./data/VBDP/trainn.csv')
    test_df = pd.read_csv('./data/VBDP/testt.csv')
else:
    train_df = pd.read_csv('./data/VBDP/train.csv')
    # remove the id column
    train_df = train_df.drop(columns=['id'])
```

```
from sklearn.preprocessing import OrdinalEncoder
n_samples = train_df.shape[0]
n_features = train_df.shape[1] - 1
target_column = "prognosis"
# Encoder our prognosis labels as integers for easier decoding later
enc = OrdinalEncoder()
train_df[target_column] = enc.fit_transform(train_df[[target_column]])
train_df.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
print(train_df.shape)
train_df.head()
```

```
(707, 65)
```

| | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | x10 | ... | x56 | x57 | x58 | x59 | x60 | x61 | x62 | x63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1.0 | 1.0 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.0 | 1.0 | 1.0 | ... | 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 | 1.0 | 0.0 | 1.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 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 | 0.0 | 1.0 | 1.0 |
| 3 | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | ... | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 | 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 | ... | x56 | x57 | x58 | x59 | x60 | x61 | x62 | x63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |
| 1 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 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 | 1.0 | 1.0 | 1.0 | 0.0 |
| 3 | 1.0 | 1.0 | 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 |
| 4 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | 1.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

```
# 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 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(),
#     )
```

## 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:

`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.

```python
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
```

```python
# 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
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:

```python
print(*fun_control["core_model_hyper_dict"].keys(), sep="\n")
```

```
C
kernel
degree
gamma
coef0
shrinking
probability
tol
cache_size
```

`break_ties`

## 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:

```
modify_hyper_parameter_bounds(fun_control, "tol", bounds=[1e-3, 1e-2])
```

```
from spotPython.hyperparameters.values import modify_hyper_parameter_bounds
modify_hyper_parameter_bounds(fun_control, "probability", bounds=[1, 1])
```

### 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 12.6.

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:

```
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
modify_hyper_parameter_levels(fun_control, "kernel", ["rbf"])
```

### 15.6.3 Optimizers

Optimizers are described in Section 12.6.1.

### 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",
})
```

#### 15.7.2.1 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,
# })
```

## 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)
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        |
```

### 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
X_start = get_default_hyperparameters_as_array(fun_control)
X_start
```

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

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

```
                    fun_control = fun_control,
                    design_control={"init_size": INIT_SIZE,
                                    "repeats": 1},
                    surrogate_control={"noise": True,
                                       "cod_type": "norm",
                                       "min_theta": -4,
                                       "max_theta": 3,
                                       "n_theta": len(var_name),
                                       "model_fun_evals": 10_000,
                                       "log_level": 50
                                       })
spot_tuner.run(X_start=X_start)
```

```
spotPython tuning: -0.38345864661654133 [----------] 0.30%

spotPython tuning: -0.38345864661654133 [----------] 0.59%

spotPython tuning: -0.38345864661654133 [----------] 0.96%

spotPython tuning: -0.38345864661654133 [----------] 1.31%

spotPython tuning: -0.38345864661654133 [----------] 1.54%

spotPython tuning: -0.38345864661654133 [----------] 1.81%

spotPython tuning: -0.38345864661654133 [----------] 2.05%

spotPython tuning: -0.38345864661654133 [----------] 2.33%

spotPython tuning: -0.38345864661654133 [----------] 2.59%

spotPython tuning: -0.39473684210526316 [----------] 2.85%

spotPython tuning: -0.39473684210526316 [----------] 3.07%

spotPython tuning: -0.39473684210526316 [----------] 3.32%

spotPython tuning: -0.39473684210526316 [----------] 3.91%
```

```
spotPython tuning: -0.39473684210526316 [----------] 4.54%

spotPython tuning: -0.39473684210526316 [#---------] 5.21%

spotPython tuning: -0.39473684210526316 [#---------] 6.10%

spotPython tuning: -0.39473684210526316 [#---------] 7.35%

spotPython tuning: -0.39473684210526316 [#---------] 8.54%

spotPython tuning: -0.39473684210526316 [#---------] 9.25%

spotPython tuning: -0.39473684210526316 [#---------] 10.08%

spotPython tuning: -0.39473684210526316 [#---------] 10.88%

spotPython tuning: -0.39473684210526316 [#---------] 11.71%

spotPython tuning: -0.39473684210526316 [#---------] 12.56%

spotPython tuning: -0.39473684210526316 [#---------] 13.48%

spotPython tuning: -0.39473684210526316 [#---------] 14.58%

spotPython tuning: -0.39473684210526316 [##--------] 15.43%

spotPython tuning: -0.39473684210526316 [##--------] 16.08%

spotPython tuning: -0.39473684210526316 [##--------] 16.94%

spotPython tuning: -0.39473684210526316 [##--------] 18.16%

spotPython tuning: -0.39473684210526316 [##--------] 19.14%

spotPython tuning: -0.39473684210526316 [##--------] 20.21%

spotPython tuning: -0.39473684210526316 [##--------] 21.67%
```

```
spotPython tuning: -0.39473684210526316 [##--------] 23.29%

spotPython tuning: -0.39473684210526316 [##--------] 24.65%

spotPython tuning: -0.39473684210526316 [###-------] 26.04%

spotPython tuning: -0.39473684210526316 [###-------] 27.90%

spotPython tuning: -0.39473684210526316 [###-------] 29.27%

spotPython tuning: -0.39473684210526316 [###-------] 30.72%

spotPython tuning: -0.39473684210526316 [###-------] 32.25%

spotPython tuning: -0.39473684210526316 [###-------] 33.78%

spotPython tuning: -0.39473684210526316 [####------] 35.13%

spotPython tuning: -0.39473684210526316 [####------] 36.41%

spotPython tuning: -0.39473684210526316 [####------] 37.61%

spotPython tuning: -0.39473684210526316 [####------] 39.58%

spotPython tuning: -0.39473684210526316 [####------] 41.07%

spotPython tuning: -0.39473684210526316 [####------] 42.39%

spotPython tuning: -0.39473684210526316 [####------] 43.72%

spotPython tuning: -0.39473684210526316 [#####-----] 45.37%

spotPython tuning: -0.39473684210526316 [#####-----] 46.64%

spotPython tuning: -0.39473684210526316 [#####-----] 48.15%

spotPython tuning: -0.39473684210526316 [#####-----] 49.61%
```

```
spotPython tuning: -0.39473684210526316 [#####-----] 51.33%

spotPython tuning: -0.39473684210526316 [#####-----] 52.87%

spotPython tuning: -0.39473684210526316 [#####-----] 54.31%

spotPython tuning: -0.39473684210526316 [######----] 55.72%

spotPython tuning: -0.39473684210526316 [######----] 57.13%

spotPython tuning: -0.39473684210526316 [######----] 58.50%

spotPython tuning: -0.39473684210526316 [######----] 60.01%

spotPython tuning: -0.39473684210526316 [#######---] 66.14%

spotPython tuning: -0.39473684210526316 [#######---] 71.83%

spotPython tuning: -0.39473684210526316 [########--] 76.90%

spotPython tuning: -0.39473684210526316 [########--] 81.26%

spotPython tuning: -0.39473684210526316 [#########-] 86.17%

spotPython tuning: -0.39473684210526316 [#########-] 91.31%

spotPython tuning: -0.39473684210526316 [##########] 95.74%

spotPython tuning: -0.39473684210526316 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x2963d3d60>
```

## 15.9 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section 12.9, 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")
```

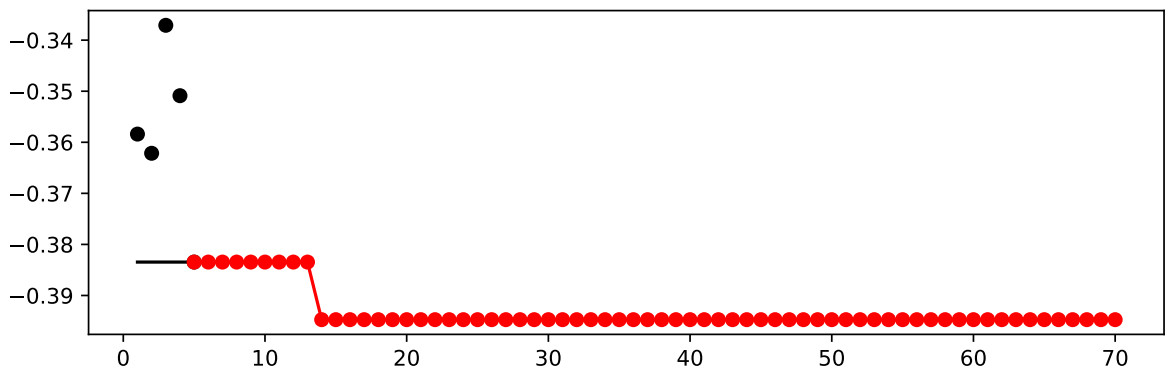



Figure 15.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- Print the results

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

```
| name        | type   | default   |   lower |   upper |                 tuned | transform
|-------------|--------|-----------|---------|---------|-----------------------|-------------
| C           | float  | 1.0       |     0.1 |    10.0 |    4.211117448021866 | None
| kernel      | factor | rbf       |     0.0 |     0.0 |                   0.0 | None
| degree      | int    | 3         |     3.0 |     3.0 |                   3.0 | None
| gamma       | factor | scale     |     0.0 |     1.0 |                   1.0 | None
| coef0       | float  | 0.0       |     0.0 |     0.0 |                   0.0 | None
| shrinking   | factor | 0         |     0.0 |     1.0 |                   1.0 | None
| probability | factor | 0         |     1.0 |     1.0 |                   1.0 | None
| tol         | float  | 0.001     |  0.0001 |    0.01 | 0.004278044656534419 | None
| cache_size  | float  | 200.0     |   100.0 |   400.0 |   319.49898598118955 | None
| break_ties  | factor | 0         |     0.0 |     1.0 |                   1.0 | None
```

### 15.10.1 Show variable importance

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

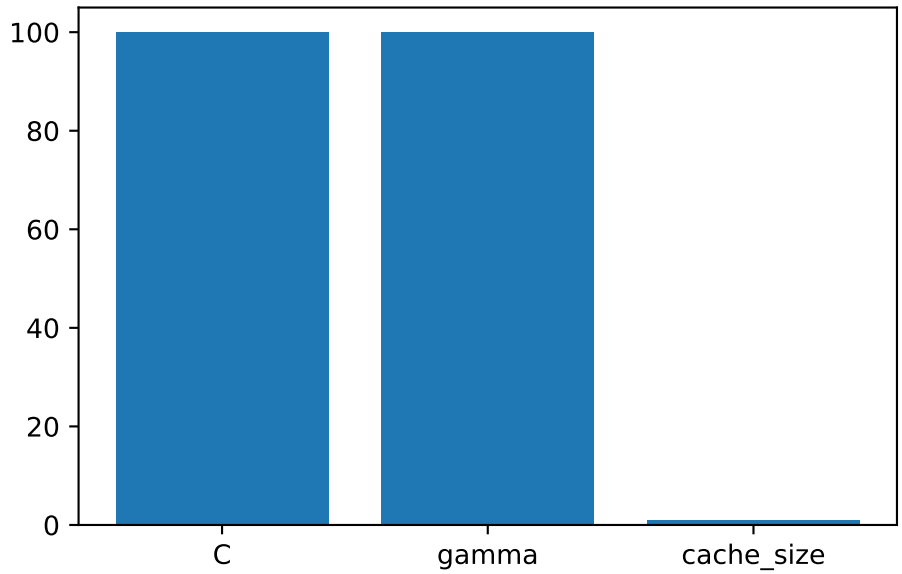



Figure 15.2: Variable importance plot, threshold 0.025.

### 15.10.2 Get Default Hyperparameters

```
from spotPython.hyperparameters.values import get_default_values, transform_hyper_paramete
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}
```

```python
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),
                ('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".
>
> ```python
> values_default.update({"probability": 1})
> ```

### 15.10.3 Get SPOT Results

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

```
[[4.21111745e+00 0.00000000e+00 3.00000000e+00 1.00000000e+00
  0.00000000e+00 1.00000000e+00 1.00000000e+00 4.27804466e-03
  3.19498986e+02 1.00000000e+00]]
```

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

```
[{'C': 4.211117448021866,
  'kernel': 'rbf',
  'degree': 3,
  'gamma': 'auto',
  'coef0': 0.0,
```

```
  'shrinking': 1,
  'probability': 1,
  'tol': 0.004278044656534419,
  'cache_size': 319.49898598118955,
  '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
```

```
SVC(C=4.211117448021866, break_ties=1, cache_size=319.49898598118955,
    gamma='auto', probability=1, shrinking=1, tol=0.004278044656534419)
```

### 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
```

```
0.3747645951035781
```

```
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-determinstic, we perform $n = 30$ runs and calculate the mean and standard deviation of the performance metric.

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

```
mean_res: 0.3769930947897049
std_res: 0.003683031459500358
min_res: 0.3700564971751412
max_res: 0.3860640301318267
median_res: 0.37664783427495285
```

### 15.10.6 Evalution 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)
```

```
0.3870056497175141
```

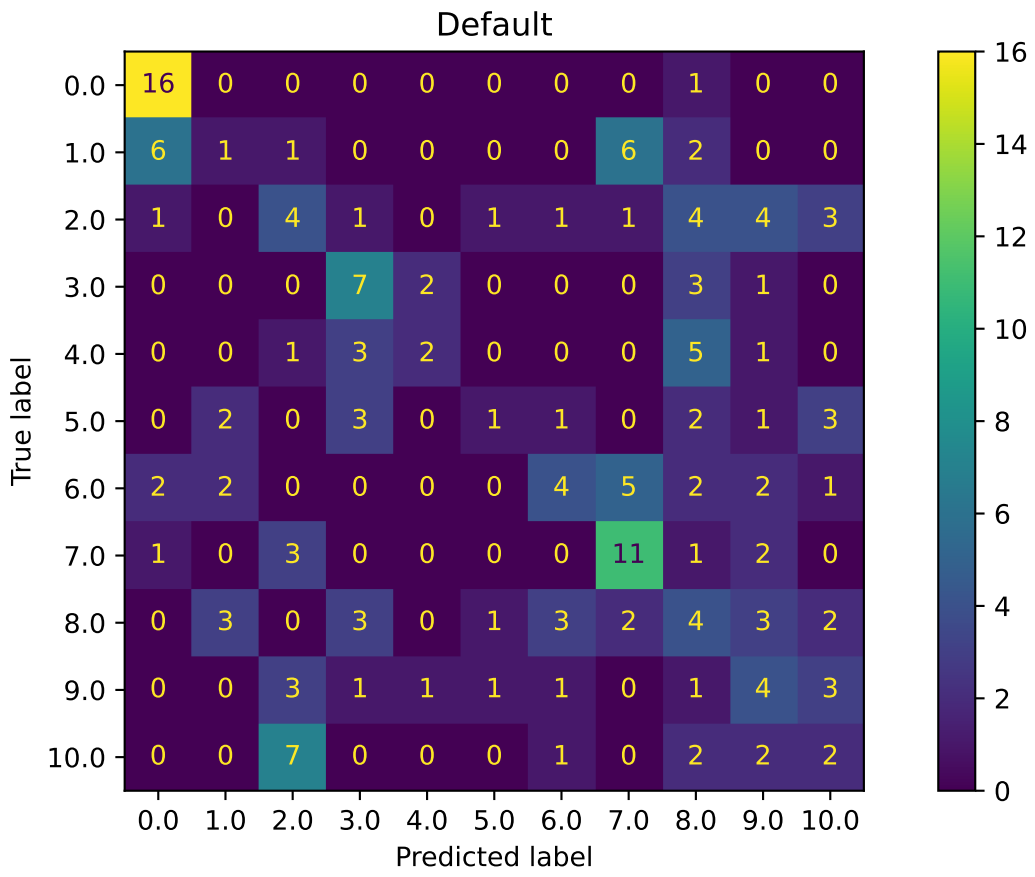
Since one single evaluation is not meaningful, we perform, similar to the evaluation of the SPOT results, $n = 30$ runs of the default setting and and calculate the mean and standard deviation of the performance metric.

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

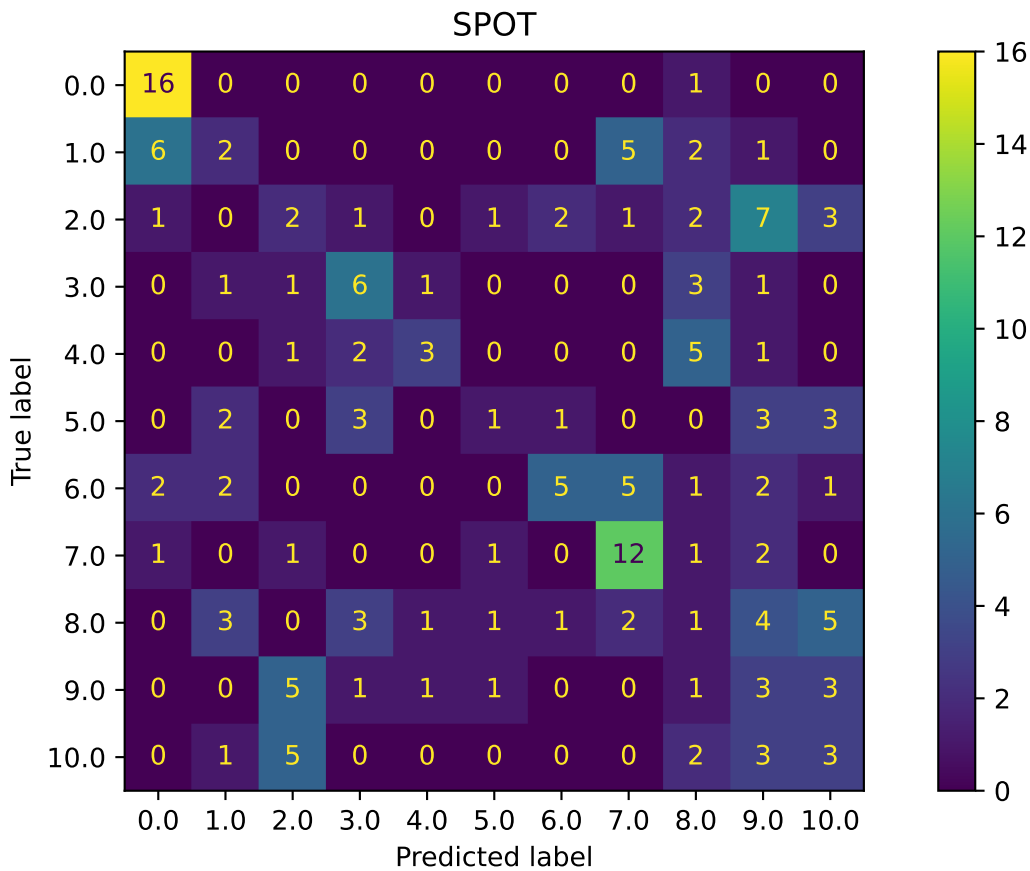
```
mean_res: 0.38471437539234155
std_res: 0.0046935326068774695
min_res: 0.37476459510357824
max_res: 0.396421845574388
median_res: 0.385593220338983
```

### 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")
```




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

```
(-0.39473684210526316, -0.3370927318295739)
```

### 15.10.8 Cross-validated Evaluations

```python
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)
```

```
(0.34874213836477985, None)
```

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

```
(0.37004357298474944, None)
```

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

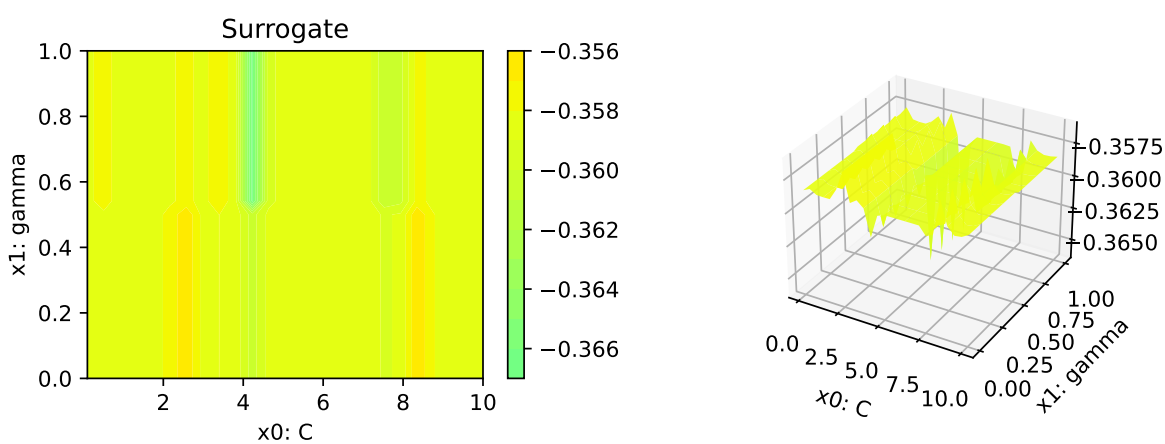
```
fun_control.update({
     "eval": "data_cv",
     "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)
```

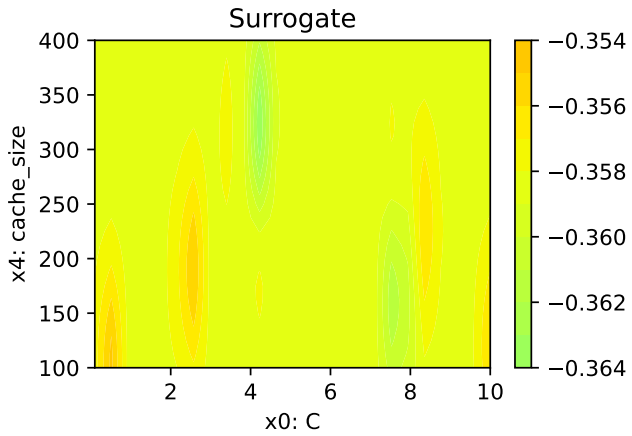
```
(0.36312541918175717, None)
```

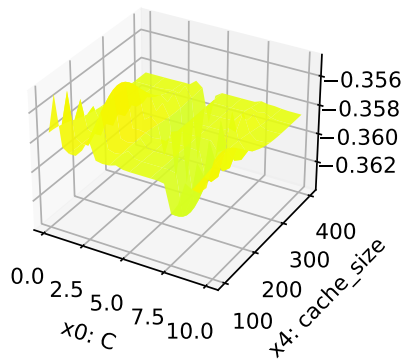
### 15.10.9 Detailed Hyperparameter Plots

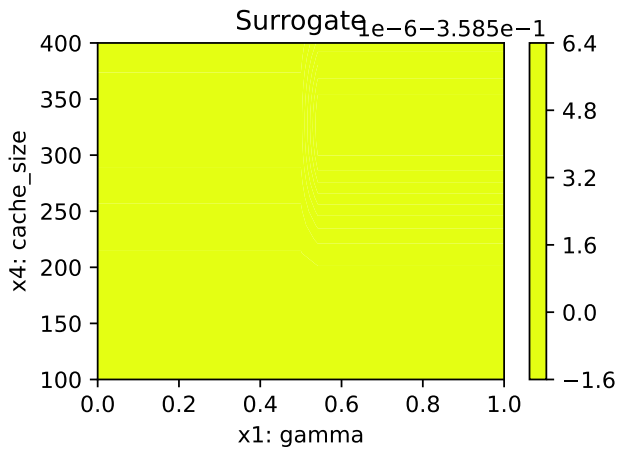
```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

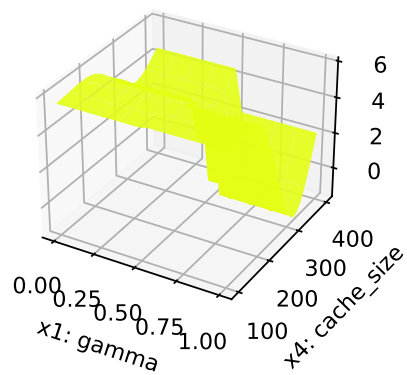
```
C:  100.0
gamma:  100.0
cache_size:  0.9390847014895988
```






### 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 KNN Classifier VBDP Data

This document refers to the following software versions:

- python: 3.10.10

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

```
spotPython                  0.3.4
spotRiver                   0.0.93

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 = '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_p040025_1min_5init_2023-07-06_02-17-50
```

```
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")
```

### 16.2.1 Load Data: Classification VBDP

```
import pandas as pd
if ORIGINAL == True:
    train_df = pd.read_csv('./data/VBDP/trainn.csv')
    test_df = pd.read_csv('./data/VBDP/testt.csv')
else:
    train_df = pd.read_csv('./data/VBDP/train.csv')
```

```
    # remove the id column
    train_df = train_df.drop(columns=['id'])
```

```
from sklearn.preprocessing import OrdinalEncoder
n_samples = train_df.shape[0]
n_features = train_df.shape[1] - 1
target_column = "prognosis"
# Encoder our prognosis labels as integers for easier decoding later
enc = OrdinalEncoder()
train_df[target_column] = enc.fit_transform(train_df[[target_column]])
train_df.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
print(train_df.shape)
train_df.head()
```

```
(707, 65)
```

| | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | x10 | ... | x56 | x57 | x58 | x59 | x60 | x61 | x62 | x63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1.0 | 1.0 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.0 | 1.0 | 1.0 | ... | 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 | 1.0 | 0.0 | 1.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 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 | 0.0 | 1.0 | 1.0 |
| 3 | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | ... | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 | 0.0 |

The full data set `train_df` 64 features. The target column is labeled as `prognosis`.

### 16.2.2 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 | ... | x56 | x57 | x58 | x59 | x60 | x61 | x62 | x63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |
| 1 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 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 | 1.0 | 1.0 | 1.0 | 0.0 |
| 3 | 1.0 | 1.0 | 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 |
| 4 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | 1.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

```
# 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})
```

## 16.3 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(),
#     )
```

## 16.4 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:

`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
```

```python
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
```

```python
# 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
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:

```python
print(*fun_control["core_model_hyper_dict"].keys(), sep="\n")
```

```
n_neighbors
weights
algorithm
leaf_size
p
```

## 16.5 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model`

### 16.5.1 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:

```
modify_hyper_parameter_bounds(fun_control, "tol", bounds=[1e-3, 1e-2])
```

```
# from spotPython.hyperparameters.values import modify_hyper_parameter_bounds
# modify_hyper_parameter_bounds(fun_control, "probability", bounds=[1, 1])
```

### 16.5.2 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 12.6.

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:

`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
# modify_hyper_parameter_levels(fun_control, "kernel", ["rbf"])
```

### 16.5.3 Optimizers

Optimizers are described in Section 12.6.1.

### 16.5.4 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.

## 16.6 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"`.

### 16.6.1 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.

#### 16.6.1.1 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}`.

#### 16.6.1.2 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},
               })
```

### 16.6.2 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",
})
```

#### 16.6.2.1 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,
# })
```

## 16.7 Step 8: Calling the SPOT Function

### 16.7.1 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)
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                  |
```

### 16.7.2 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.hypersklearn import HyperSklearn
fun = HyperSklearn().fun_sklearn
```

### 16.7.3 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: `initi_size`, 20 points) is not considered.

```
from spotPython.hyperparameters.values import get_default_hyperparameters_as_array
X_start = get_default_hyperparameters_as_array(fun_control)
X_start
```

```
array([[2, 0, 0, 5, 2]])
```

```python
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 [----------] 0.24%

spotPython tuning: -0.3107769423558897 [----------] 0.52%

spotPython tuning: -0.3107769423558897 [----------] 0.78%

spotPython tuning: -0.3107769423558897 [----------] 1.04%

spotPython tuning: -0.3107769423558897 [----------] 1.30%
```

```
spotPython tuning: -0.3107769423558897 [----------] 1.62%

spotPython tuning: -0.3107769423558897 [----------] 1.98%

spotPython tuning: -0.3107769423558897 [----------] 2.31%

spotPython tuning: -0.3107769423558897 [----------] 2.64%

spotPython tuning: -0.3107769423558897 [----------] 2.93%

spotPython tuning: -0.3107769423558897 [----------] 3.22%

spotPython tuning: -0.3107769423558897 [----------] 4.14%

spotPython tuning: -0.3107769423558897 [#---------] 5.08%

spotPython tuning: -0.3107769423558897 [#---------] 6.23%

spotPython tuning: -0.3107769423558897 [#---------] 7.35%

spotPython tuning: -0.3107769423558897 [#---------] 8.53%

spotPython tuning: -0.3107769423558897 [#---------] 9.90%

spotPython tuning: -0.3107769423558897 [#---------] 10.96%

spotPython tuning: -0.3107769423558897 [#---------] 12.86%

spotPython tuning: -0.3107769423558897 [#---------] 14.07%

spotPython tuning: -0.3107769423558897 [##--------] 15.07%

spotPython tuning: -0.3107769423558897 [##--------] 15.90%

spotPython tuning: -0.3107769423558897 [##--------] 16.84%

spotPython tuning: -0.3107769423558897 [##--------] 17.97%
```

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

spotPython tuning: -0.3107769423558897 [##--------] 20.02%

spotPython tuning: -0.3107769423558897 [##--------] 21.36%

spotPython tuning: -0.3107769423558897 [##--------] 22.87%

spotPython tuning: -0.3107769423558897 [##--------] 24.58%

spotPython tuning: -0.3107769423558897 [###-------] 26.46%

spotPython tuning: -0.3107769423558897 [###-------] 28.16%

spotPython tuning: -0.3107769423558897 [###-------] 30.65%

spotPython tuning: -0.3107769423558897 [###-------] 32.70%

spotPython tuning: -0.3107769423558897 [####------] 35.23%

spotPython tuning: -0.3107769423558897 [####------] 38.99%

spotPython tuning: -0.3107769423558897 [####------] 41.80%

spotPython tuning: -0.3107769423558897 [####------] 44.46%

spotPython tuning: -0.3107769423558897 [#####-----] 47.34%

spotPython tuning: -0.3107769423558897 [#####-----] 50.02%

spotPython tuning: -0.3107769423558897 [#####-----] 52.79%

spotPython tuning: -0.3107769423558897 [######----] 55.92%

spotPython tuning: -0.3107769423558897 [######----] 59.03%

spotPython tuning: -0.3107769423558897 [######----] 62.37%
```

```
spotPython tuning: -0.3107769423558897 [#######---] 65.50%

spotPython tuning: -0.3107769423558897 [#######---] 68.26%

spotPython tuning: -0.3107769423558897 [#######---] 71.23%

spotPython tuning: -0.3107769423558897 [########--] 75.27%

spotPython tuning: -0.3107769423558897 [########--] 78.38%

spotPython tuning: -0.3107769423558897 [########--] 81.61%

spotPython tuning: -0.3107769423558897 [#########-] 85.21%

spotPython tuning: -0.3107769423558897 [#########-] 89.23%

spotPython tuning: -0.3107769423558897 [#########-] 92.65%

spotPython tuning: -0.3107769423558897 [##########] 95.24%

spotPython tuning: -0.3107769423558897 [##########] 98.34%

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

<spotPython.spot.spot.Spot at 0x1479b6bc0>
```

## 16.8 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section 12.9, see also the description in the documentation: Tensorboard.

## 16.9 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")
```

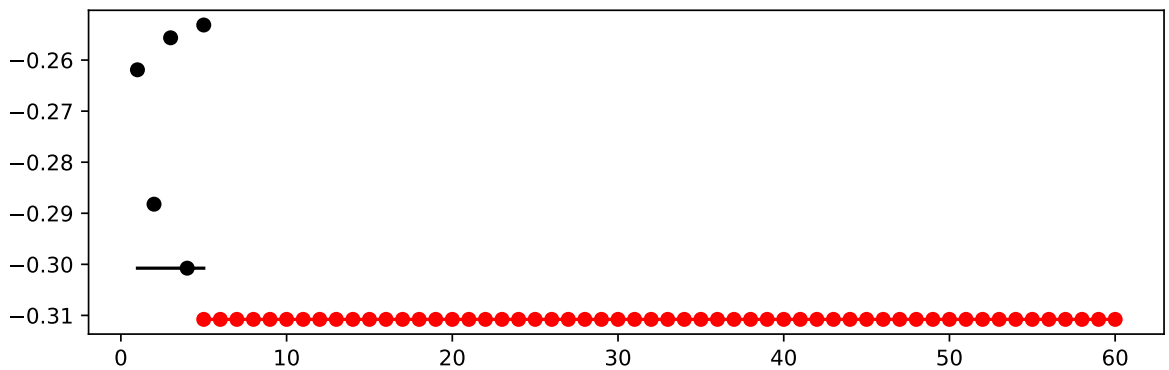



Figure 16.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- Print the results

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

```
| name        | type   | default   |   lower |   upper |   tuned | 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                  |
```

### 16.9.1 Show variable importance

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

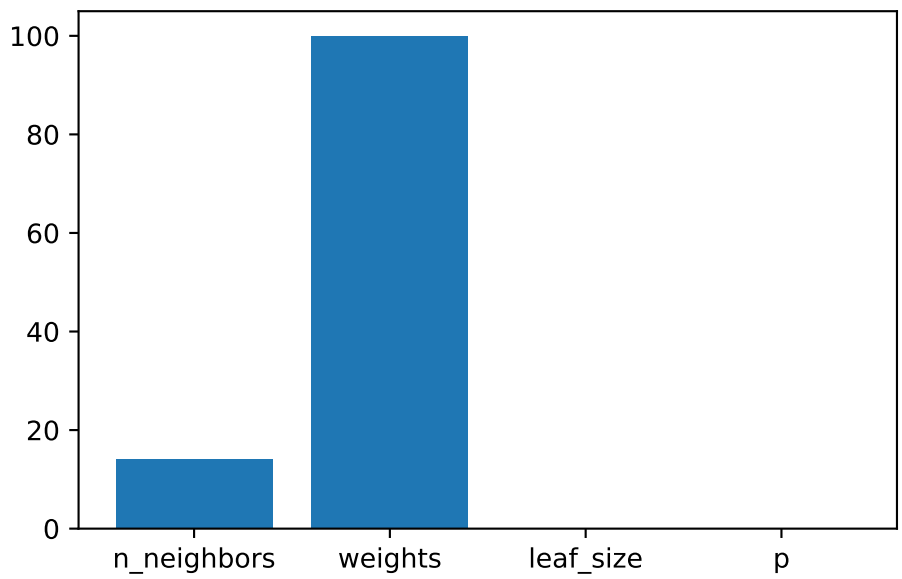



Figure 16.2: Variable importance plot, threshold 0.025.

### 16.9.2 Get Default Hyperparameters

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

```
Pipeline(steps=[('nonetype', None),
                ('kneighborsclassifier',
                 KNeighborsClassifier(leaf_size=32, n_neighbors=4))])
```

### 16.9.3 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_dic
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')
```

### 16.9.4 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
```

### 16.9.5 Handling Non-deterministic Results

- Because the model is non-determinstic, 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
```

### 16.9.6 Evalution 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
```

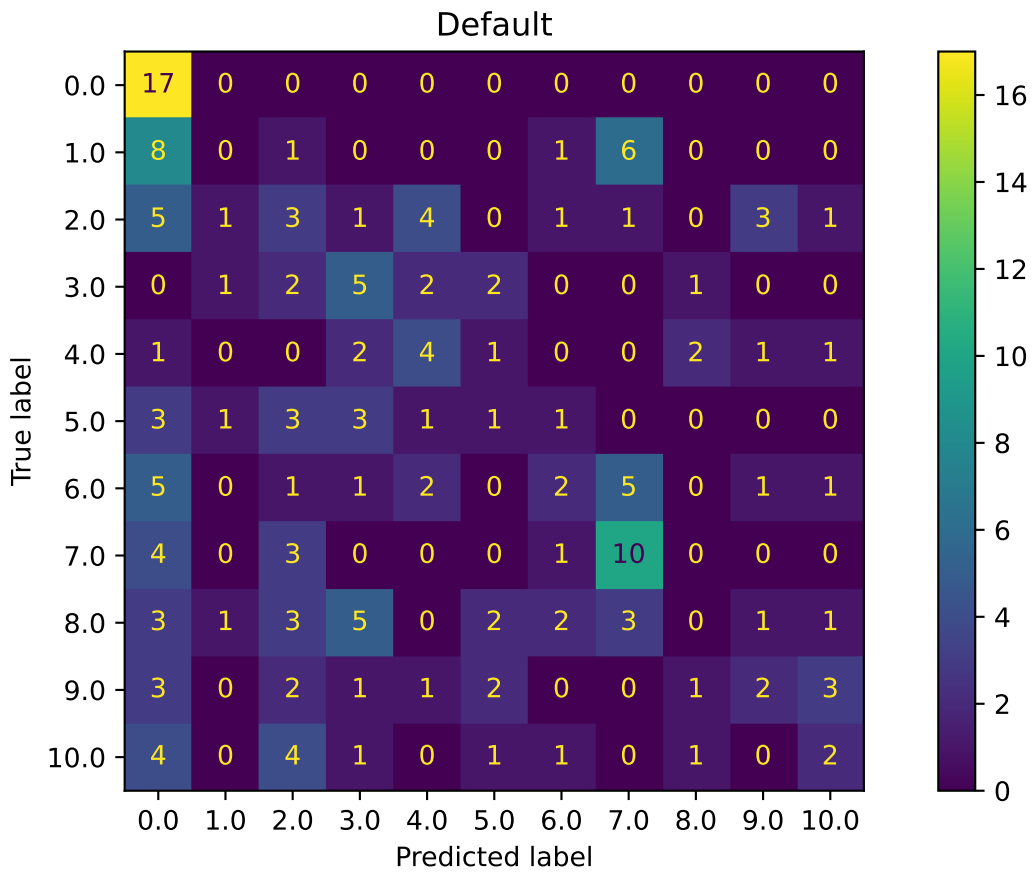
Since one single evaluation is not meaningful, we perform, similar to the evaluation of the SPOT results, $n = 30$ runs of the default setting and and calculate the mean and standard deviation of the performance metric.

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

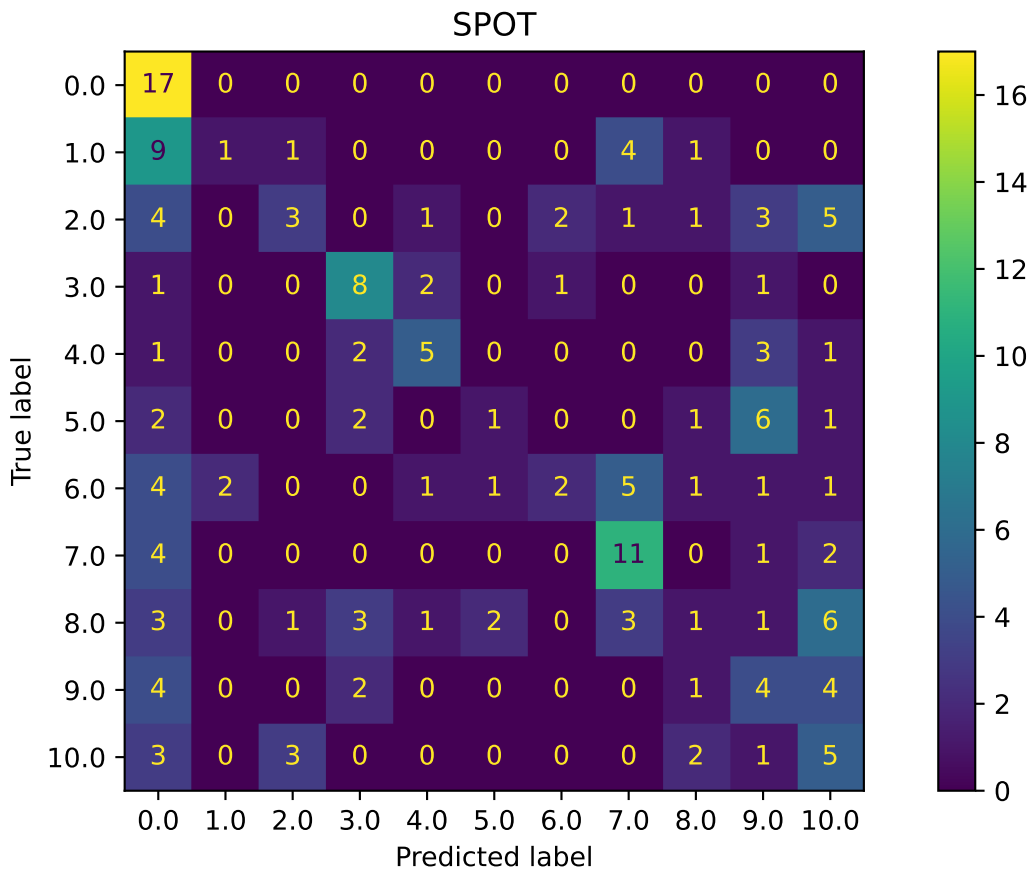
```
mean_res: 0.2768361581920903
std_res: 1.1102230246251565e-16
min_res: 0.2768361581920904
max_res: 0.2768361581920904
median_res: 0.2768361581920904
```

### 16.9.7 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)
```

### 16.9.8 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:

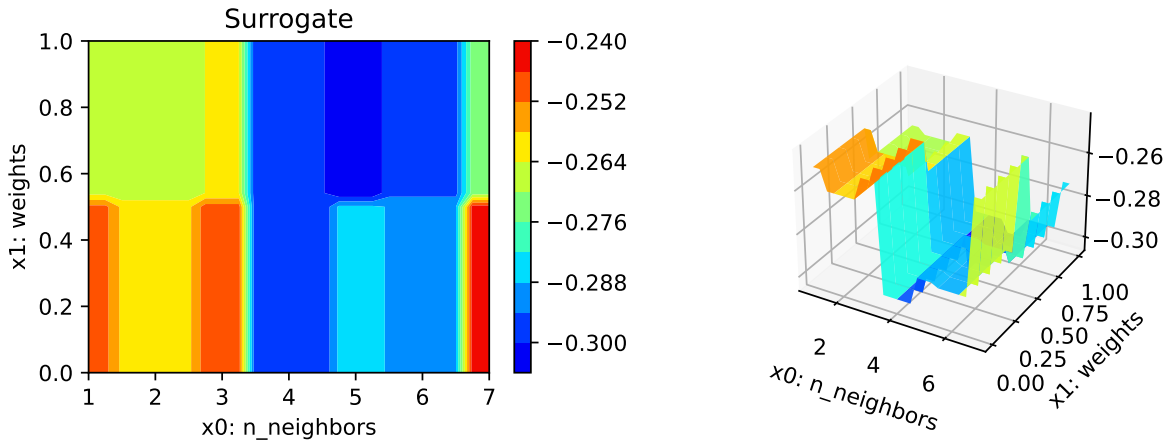
```
fun_control.update({
     "eval": "data_cv",
     "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)
```

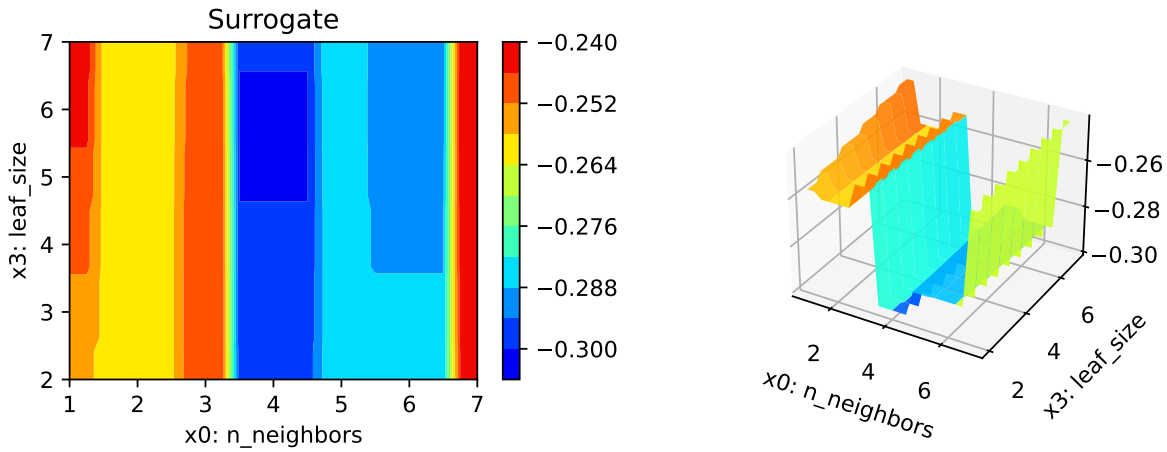
```
(0.3061904761904762, None)
```

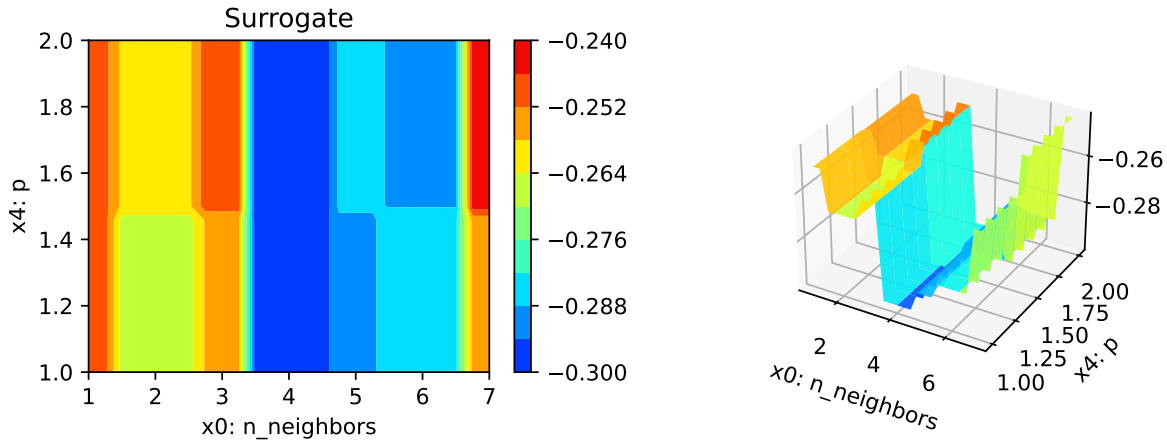
### 16.9.9 Detailed Hyperparameter Plots

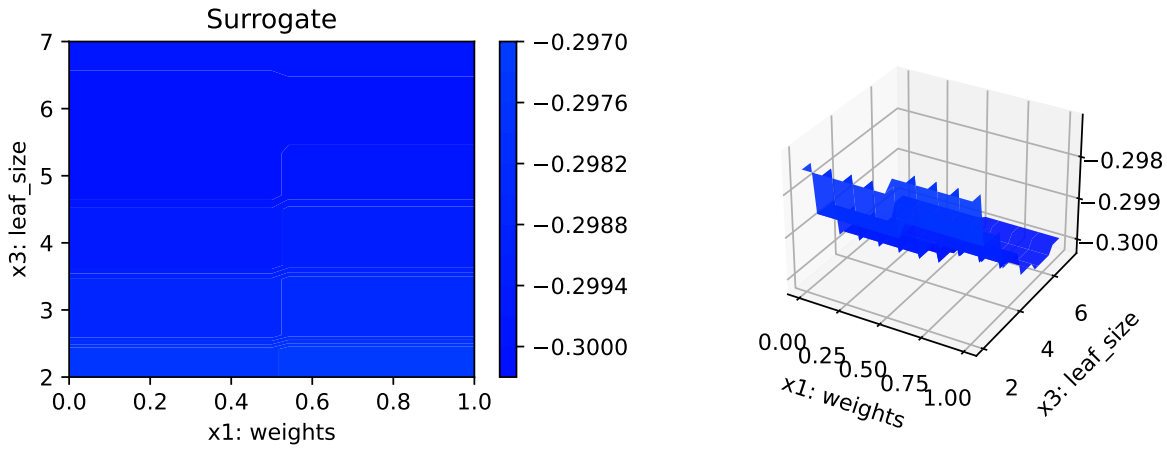
```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

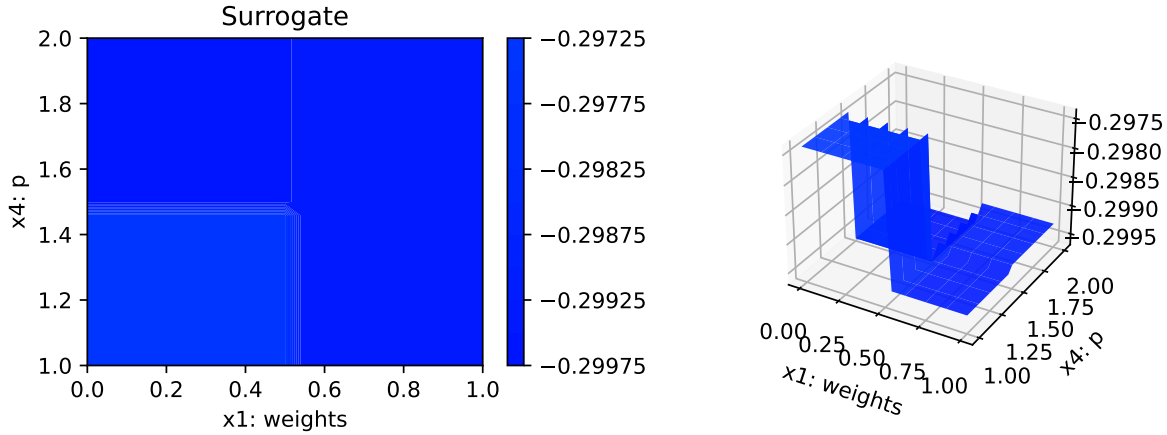
```
n_neighbors:  14.062005356823159
weights:  100.0
leaf_size:  0.03194210846917897
p:  0.026575949846363224
```

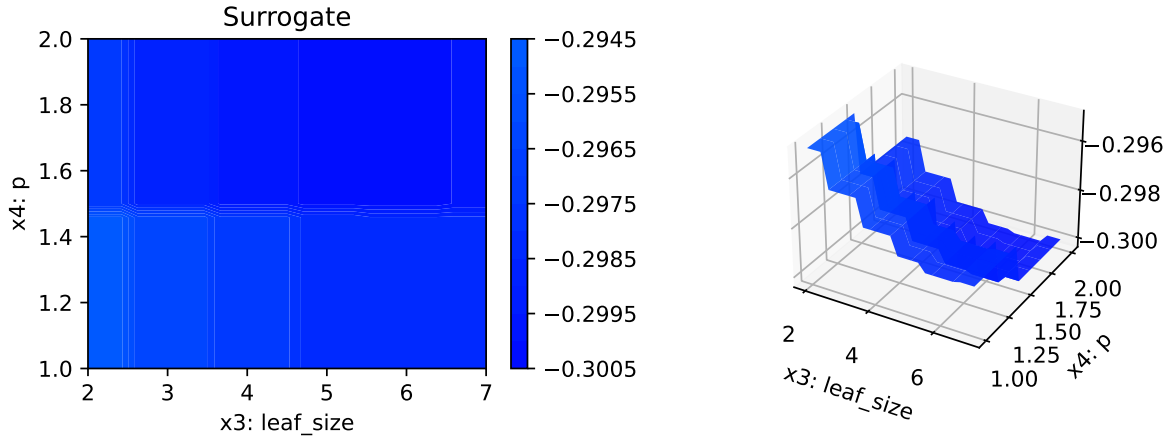

Surrogate
x1: weights
x0: n_neighbors
−0.240
−0.252
−0.264
−0.276
−0.288
−0.300
x0: n_neighbors
x1: weights


Surrogate
x3: leaf_size
x0: n_neighbors
−0.240
−0.252
−0.264
−0.276
−0.288
−0.300
x0: n_neighbors
x3: leaf_size


Surrogate
x4: p
x0: n_neighbors
−0.240
−0.252
−0.264
−0.276
−0.288
−0.300
x0: n_neighbors
x4: p


Surrogate
x3: leaf_size
x1: weights
−0.2970
−0.2976
−0.2982
−0.2988
−0.2994
−0.3000
−0.298
−0.299
−0.300
x1: weights
x3: leaf_size


Surrogate
x4: p
x1: weights
−0.29725
−0.29775
−0.29825
−0.29875
−0.29925
−0.29975
−0.2975
−0.2980
−0.2985
−0.2990
−0.2995
x1: weights
x4: p


Surrogate
x4: p
x3: leaf_size
−0.2945
−0.2955
−0.2965
−0.2975
−0.2985
−0.2995
−0.3005
−0.296
−0.298
−0.300
x3: leaf_size
x4: p

### 16.9.10 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
```

### 16.9.11 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)
```

# 17 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 latest `spotPython` version, which can be installed via pip. Alternatively, the source code can be downloaded from gitHub: https://github.com/sequential-parameter-optimization/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, initial design size, etc.
- The parameter `MAX_TIME` specifies the maximum run time in seconds.
- The parameter `INIT_SIZE` specifies the initial design size.
- The parameter `WORKERS` specifies the number of workers.
- The prefix `PREFIX` is used for the experiment name and the name of the log file.

```
MAX_TIME = 1
INIT_SIZE = 5
WORKERS = 0
PREFIX="31"
```

```
import os
if not os.path.exists('./figures'):
    os.makedirs('./figures')
```

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

- Although there are no .cuda() or .to(device) calls required, because Lightning does these for you, see LIGHTNINGMODULE, we would like to know which device is used. Threrefore, we imitate the LightningModule behaviour which selects the highest device.
- The method `spotPython.utils.device.getDevice()` returns the device that is used by Lightning.

## 17.2 Step 2: Initialization of the `fun_control` Dictionary

`spotPython` uses a Python dictionary for storing the information required for the hyperparameter tuning process, which was described in Section 12.2, 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
from spotPython.utils.device import getDevice

experiment_name = get_experiment_name(prefix=PREFIX)
fun_control = fun_control_init(
    num_workers=WORKERS,
    device=getDevice(),
    _L_in=64,
    _L_out=11)
```

```
fun_control["device"]
```

'mps'

## 17.3 Step 3: PyTorch Data Loading

### 17.3.1 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]

Section Section 17.13.1 illustrates how to access the data.

## 17.4 Step 4: Preprocessing

Preprocessing is handled by `Lightning` and `PyTorch`. It can be implemented in the `CSVDataModule` class [SOURCE] and is described in the LIGHTNINGDATAMODULE documentation. Here you can find information about the `transforms` methods.

## 17.5 Step 5: Select the NN Model (`algorithm`) and `core_model_hyper_dict`

`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.

- Here we simply add the NN Model to the fun_control dictionary by calling the function `add_core_model_to_fun_control`:

```
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
```

```
add_core_model_to_fun_control(core_model=NetLightBase,
                              fun_control=fun_control,
                              hyper_dict= LightHyperDict)
```

The `NetLightBase` is a configurable neural network. The hyperparameters of the model are specified in the `core_model_hyper_dict` dictionary [SOURCE].

## 17.6 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 12.6.

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

```
from spotPython.hyperparameters.values import modify_hyper_parameter_bounds

modify_hyper_parameter_bounds(fun_control, "l1", bounds=[6,13])
modify_hyper_parameter_bounds(fun_control, "epochs", bounds=[6,13])
modify_hyper_parameter_bounds(fun_control, "batch_size", bounds=[2, 8])
```

```
from spotPython.hyperparameters.values import modify_hyper_parameter_levels
modify_hyper_parameter_levels(fun_control, "optimizer",["Adam", "AdamW", "Adamax", "NAdam"
# modify_hyper_parameter_levels(fun_control, "optimizer", ["Adam"])
```

The updated `fun_control` dictionary is shown below.

```
fun_control["core_model_hyper_dict"]
```

```
{'l1': {'type': 'int',
```

```
 'default': 3,
 'transform': 'transform_power_2_int',
 'lower': 6,
 'upper': 13},
'epochs': {'type': 'int',
 'default': 4,
 'transform': 'transform_power_2_int',
 'lower': 6,
 'upper': 13},
'batch_size': {'type': 'int',
 'default': 4,
 'transform': 'transform_power_2_int',
 'lower': 2,
 'upper': 8},
'act_fn': {'levels': ['Sigmoid', 'Tanh', 'ReLU', 'LeakyReLU', 'ELU', 'Swish'],
 'type': 'factor',
 'default': 'ReLU',
 'transform': 'None',
 'class_name': 'spotPython.torch.activation',
 'core_model_parameter_type': 'instance()',
 'lower': 0,
 'upper': 5},
'optimizer': {'levels': ['Adam', 'AdamW', 'Adamax', 'NAdam'],
 'type': 'factor',
 'default': 'SGD',
 'transform': 'None',
 'class_name': 'torch.optim',
 'core_model_parameter_type': 'str',
 'lower': 0,
 'upper': 3},
'dropout_prob': {'type': 'float',
 'default': 0.01,
 'transform': 'None',
 'lower': 0.0,
 'upper': 0.25},
'lr_mult': {'type': 'float',
 'default': 1.0,
 'transform': 'None',
 'lower': 0.1,
 'upper': 10.0},
'patience': {'type': 'int',
 'default': 2,
 'transform': 'transform_power_2_int',
```

```
  'lower': 2,
  'upper': 6},
 'initialization': {'levels': ['Default', 'Kaiming', 'Xavier'],
  'type': 'factor',
  'default': 'Default',
  'transform': 'None',
  'core_model_parameter_type': 'str',
  'lower': 0,
  'upper': 2}}
```

## 17.7 Step 7: Data Splitting, the Objective (Loss) Function and the Metric

### 17.7.1 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 12.7.1)
2. the loss function (and a metric).

> Caution: Data Splitting in Lightning
>
> - The data splitting is handled by `Lightning`.

### 17.7.2 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.

### 17.7.3 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`.

## 17.8 Step 8: Calling the SPOT Function

### 17.8.1 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)
lower = get_bound_values(fun_control, "lower")
upper = get_bound_values(fun_control, "upper")
```

Now, the dictionary `fun_control` contains all information needed for the hyperparameter tuning. Before the hyperparameter tuning is started, it is recommended to take a look at the experimental design. The method `gen_design_table` [SOURCE] generates a design table as follows:

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

```
| name           | type   | default   |   lower |   upper | transform             |
|----------------|--------|-----------|---------|---------|-----------------------|
| l1             | int    | 3         |     6   |   13    | transform_power_2_int |
| epochs         | int    | 4         |     6   |   13    | transform_power_2_int |
| batch_size     | int    | 4         |     2   |    8    | transform_power_2_int |
| act_fn         | factor | ReLU      |     0   |    5    | None                  |
| optimizer      | factor | SGD       |     0   |    3    | None                  |
| dropout_prob   | float  | 0.01      |     0   |    0.25 | None                  |
| lr_mult        | float  | 1.0       |     0.1 |   10    | None                  |
| patience       | int    | 2         |     2   |    6    | transform_power_2_int |
| initialization | factor | Default   |     0   |    2    | None                  |
```

This allows to check if all information is available and if the information is correct.

### 17.8.2 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.fun.hyperlight import HyperLight
fun = HyperLight().fun
```

### 17.8.3 Starting the Hyperparameter Tuning

The `spotPython` hyperparameter tuning is started by calling the `Spot` function [SOURCE] as described in Section 12.8.4.

```
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,
                   max_time = MAX_TIME,
                   tolerance_x = np.sqrt(np.spacing(1)),
```

```
                    var_type = var_type,
                    var_name = var_name,
                    show_progress= True,
                    fun_control = fun_control,
                    design_control={"init_size": INIT_SIZE},
                    surrogate_control={"noise": True,
                                       "min_theta": -4,
                                       "max_theta": 3,
                                       "n_theta": len(var_name),
                                       "model_fun_evals": 10_000,
                                       })
spot_tuner.run()
```

```
config: {'l1': 4096, 'epochs': 4096, 'batch_size': 32, 'act_fn': ReLU(), 'optimizer': 'AdamW
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=4096, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.10939527466721133, inplace=False)
    (3): Linear(in_features=4096, out_features=2048, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.10939527466721133, inplace=False)
    (6): Linear(in_features=2048, out_features=2048, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.10939527466721133, inplace=False)
    (9): Linear(in_features=2048, out_features=1024, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.10939527466721133, inplace=False)
    (12): Linear(in_features=1024, out_features=11, bias=True)
  )
)
```

```
      Validate metric              DataLoader 0
```

```
         hp_metric              2.458235025405884
          val_acc              0.08480565249919891
         val_loss               2.458235025405884
        valid_mapk             0.1667952686548233


train_model result: {'valid_mapk': 0.1667952686548233, 'val_loss': 2.458235025405884, 'val_ac

config: {'l1': 64, 'epochs': 128, 'batch_size': 256, 'act_fn': LeakyReLU(), 'optimizer': 'Ada
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): LeakyReLU()
    (2): Dropout(p=0.012926647388264517, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): LeakyReLU()
    (5): Dropout(p=0.012926647388264517, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): LeakyReLU()
    (8): Dropout(p=0.012926647388264517, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): LeakyReLU()
    (11): Dropout(p=0.012926647388264517, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)


       Validate metric              DataLoader 0

         hp_metric              2.2916417121887207
          val_acc               0.24028268456459045
         val_loss               2.2916417121887207
        valid_mapk              0.36009839177131653


train_model result: {'valid_mapk': 0.36009839177131653, 'val_loss': 2.2916417121887207, 'val_
```

```
config: {'l1': 1024, 'epochs': 256, 'batch_size': 8, 'act_fn': Swish(), 'optimizer': 'NAdam'
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=1024, bias=True)
    (1): Swish()
    (2): Dropout(p=0.22086376796923401, inplace=False)
    (3): Linear(in_features=1024, out_features=512, bias=True)
    (4): Swish()
    (5): Dropout(p=0.22086376796923401, inplace=False)
    (6): Linear(in_features=512, out_features=512, bias=True)
    (7): Swish()
    (8): Dropout(p=0.22086376796923401, inplace=False)
    (9): Linear(in_features=512, out_features=256, bias=True)
    (10): Swish()
    (11): Dropout(p=0.22086376796923401, inplace=False)
    (12): Linear(in_features=256, out_features=11, bias=True)
  )
)
```

| Validate metric | DataLoader 0 |
|---|---|
| hp_metric | 2.451167345046997 |
| val_acc | 0.09187278896570206 |
| val_loss | 2.451167345046997 |
| valid_mapk | 0.15740738809108734 |

```
train_model result: {'valid_mapk': 0.15740738809108734, 'val_loss': 2.451167345046997, 'val_a

config: {'l1': 512, 'epochs': 512, 'batch_size': 16, 'act_fn': Sigmoid(), 'optimizer': 'Adam
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
```

```
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=512, bias=True)
    (1): Sigmoid()
    (2): Dropout(p=0.1890928563375006, inplace=False)
    (3): Linear(in_features=512, out_features=256, bias=True)
    (4): Sigmoid()
    (5): Dropout(p=0.1890928563375006, inplace=False)
    (6): Linear(in_features=256, out_features=256, bias=True)
    (7): Sigmoid()
    (8): Dropout(p=0.1890928563375006, inplace=False)
    (9): Linear(in_features=256, out_features=128, bias=True)
    (10): Sigmoid()
    (11): Dropout(p=0.1890928563375006, inplace=False)
    (12): Linear(in_features=128, out_features=11, bias=True)
  )
)
```

| Validate metric | DataLoader 0 |
|---|---|
| hp_metric | 2.2906553745269775 |
| val_acc | 0.25088340044021606 |
| val_loss | 2.2906553745269775 |
| valid_mapk | 0.3221275210380554 |

```
train_model result: {'valid_mapk': 0.3221275210380554, 'val_loss': 2.2906553745269775, 'val_a

config: {'l1': 256, 'epochs': 4096, 'batch_size': 64, 'act_fn': ReLU(), 'optimizer': 'Adamax
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=256, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.0708380794924471, inplace=False)
    (3): Linear(in_features=256, out_features=128, bias=True)
    (4): ReLU()
```

```
    (5): Dropout(p=0.0708380794924471, inplace=False)
    (6): Linear(in_features=128, out_features=128, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.0708380794924471, inplace=False)
    (9): Linear(in_features=128, out_features=64, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.0708380794924471, inplace=False)
    (12): Linear(in_features=64, out_features=11, bias=True)
  )
)


       Validate metric              DataLoader 0

          hp_metric              2.2840261459350586
           val_acc               0.25441697239875793
          val_loss               2.2840261459350586
         valid_mapk              0.34810957312583923


train_model result: {'valid_mapk': 0.34810957312583923, 'val_loss': 2.2840261459350586, 'val_

config: {'l1': 2048, 'epochs': 8192, 'batch_size': 8, 'act_fn': LeakyReLU(), 'optimizer': 'Ad
_L_in: 64
_L_out: 11

model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=2048, bias=True)
    (1): LeakyReLU()
    (2): Dropout(p=0.25, inplace=False)
    (3): Linear(in_features=2048, out_features=1024, bias=True)
    (4): LeakyReLU()
    (5): Dropout(p=0.25, inplace=False)
    (6): Linear(in_features=1024, out_features=1024, bias=True)
    (7): LeakyReLU()
    (8): Dropout(p=0.25, inplace=False)
    (9): Linear(in_features=1024, out_features=512, bias=True)
    (10): LeakyReLU()
```

```
    (11): Dropout(p=0.25, inplace=False)
    (12): Linear(in_features=512, out_features=11, bias=True)
  )
)


       Validate metric              DataLoader 0

          hp_metric               2.451167345046997
           val_acc               0.09187278896570206
          val_loss                2.451167345046997
         valid_mapk              0.16030091047286987


train_model result: {'valid_mapk': 0.16030091047286987, 'val_loss': 2.451167345046997, 'val_a

spotPython tuning: 2.2840261459350586 [#---------] 8.70%


config: {'l1': 128, 'epochs': 2048, 'batch_size': 64, 'act_fn': ReLU(), 'optimizer': 'AdamW'
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=128, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.06509509251393066, inplace=False)
    (3): Linear(in_features=128, out_features=64, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.06509509251393066, inplace=False)
    (6): Linear(in_features=64, out_features=64, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.06509509251393066, inplace=False)
    (9): Linear(in_features=64, out_features=32, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.06509509251393066, inplace=False)
    (12): Linear(in_features=32, out_features=11, bias=True)
  )
)
```

```
       Validate metric              DataLoader 0

          hp_metric              2.2531075477600098
           val_acc               0.28975266218185425
          val_loss               2.2531075477600098
         valid_mapk              0.3932677209377289


train_model result: {'valid_mapk': 0.3932677209377289, 'val_loss': 2.2531075477600098, 'val_a

spotPython tuning: 2.2531075477600098 [#---------] 11.37%


config: {'l1': 64, 'epochs': 128, 'batch_size': 64, 'act_fn': Tanh(), 'optimizer': 'AdamW',
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.038283159845823055, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.038283159845823055, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.038283159845823055, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.038283159845823055, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)


       Validate metric              DataLoader 0
```

```
         hp_metric              2.282576560974121
          val_acc              0.25088340044021606
         val_loss               2.282576560974121
        valid_mapk             0.3321759104728699


train_model result: {'valid_mapk': 0.3321759104728699, 'val_loss': 2.282576560974121, 'val_ac

spotPython tuning: 2.2531075477600098 [##--------] 15.77%


config: {'l1': 256, 'epochs': 2048, 'batch_size': 64, 'act_fn': LeakyReLU(), 'optimizer': 'Ad
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=256, bias=True)
    (1): LeakyReLU()
    (2): Dropout(p=0.13697485711027166, inplace=False)
    (3): Linear(in_features=256, out_features=128, bias=True)
    (4): LeakyReLU()
    (5): Dropout(p=0.13697485711027166, inplace=False)
    (6): Linear(in_features=128, out_features=128, bias=True)
    (7): LeakyReLU()
    (8): Dropout(p=0.13697485711027166, inplace=False)
    (9): Linear(in_features=128, out_features=64, bias=True)
    (10): LeakyReLU()
    (11): Dropout(p=0.13697485711027166, inplace=False)
    (12): Linear(in_features=64, out_features=11, bias=True)
  )
)


       Validate metric              DataLoader 0

         hp_metric              2.300975799560547
          val_acc              0.2332155406475067
         val_loss               2.300975799560547
```

```
        valid_mapk              0.3299768567085266


train_model result: {'valid_mapk': 0.3299768567085266, 'val_loss': 2.300975799560547, 'val_ac

spotPython tuning: 2.2531075477600098 [##--------] 18.76%


config: {'l1': 128, 'epochs': 256, 'batch_size': 32, 'act_fn': LeakyReLU(), 'optimizer': 'Ada
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=128, bias=True)
    (1): LeakyReLU()
    (2): Dropout(p=0.0, inplace=False)
    (3): Linear(in_features=128, out_features=64, bias=True)
    (4): LeakyReLU()
    (5): Dropout(p=0.0, inplace=False)
    (6): Linear(in_features=64, out_features=64, bias=True)
    (7): LeakyReLU()
    (8): Dropout(p=0.0, inplace=False)
    (9): Linear(in_features=64, out_features=32, bias=True)
    (10): LeakyReLU()
    (11): Dropout(p=0.0, inplace=False)
    (12): Linear(in_features=32, out_features=11, bias=True)
  )
)


       Validate metric              DataLoader 0

          hp_metric              2.2701234817504883
           val_acc               0.2614840865135193
          val_loss               2.2701234817504883
         valid_mapk              0.3686985373497009


train_model result: {'valid_mapk': 0.3686985373497009, 'val_loss': 2.2701234817504883, 'val_a
```

```
spotPython tuning: 2.2531075477600098 [###-------] 25.69%


config: {'l1': 128, 'epochs': 128, 'batch_size': 16, 'act_fn': ReLU(), 'optimizer': 'NAdam',
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=128, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.25, inplace=False)
    (3): Linear(in_features=128, out_features=64, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.25, inplace=False)
    (6): Linear(in_features=64, out_features=64, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.25, inplace=False)
    (9): Linear(in_features=64, out_features=32, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.25, inplace=False)
    (12): Linear(in_features=32, out_features=11, bias=True)
  )
)


       Validate metric              DataLoader 0

          hp_metric              2.3522214889526367
           val_acc               0.19081272184848785
          val_loss               2.3522214889526367
         valid_mapk              0.2594697177410126


train_model result: {'valid_mapk': 0.2594697177410126, 'val_loss': 2.3522214889526367, 'val_a

spotPython tuning: 2.2531075477600098 [####------] 35.50%
```

```
config: {'l1': 64, 'epochs': 2048, 'batch_size': 16, 'act_fn': ReLU(), 'optimizer': 'AdamW',
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.0, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.0, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.0, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.0, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)


       Validate metric              DataLoader 0

          hp_metric              2.3129701614379883
           val_acc               0.20494699478149414
          val_loss               2.3129701614379883
         valid_mapk              0.2976641356945038


train_model result: {'valid_mapk': 0.2976641356945038, 'val_loss': 2.3129701614379883, 'val_a

spotPython tuning: 2.2531075477600098 [####------] 40.37%


config: {'l1': 128, 'epochs': 512, 'batch_size': 256, 'act_fn': ReLU(), 'optimizer': 'Adamax
_L_in: 64
```

```
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=128, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.0559729574420195, inplace=False)
    (3): Linear(in_features=128, out_features=64, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.0559729574420195, inplace=False)
    (6): Linear(in_features=64, out_features=64, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.0559729574420195, inplace=False)
    (9): Linear(in_features=64, out_features=32, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.0559729574420195, inplace=False)
    (12): Linear(in_features=32, out_features=11, bias=True)
  )
)


       Validate metric              DataLoader 0

          hp_metric              2.2588024139404297
           val_acc               0.2720848023891449
          val_loss               2.2588024139404297
         valid_mapk              0.34910300374031067


train_model result: {'valid_mapk': 0.34910300374031067, 'val_loss': 2.2588024139404297, 'val_

spotPython tuning: 2.2531075477600098 [####------] 44.77%


config: {'l1': 128, 'epochs': 4096, 'batch_size': 8, 'act_fn': LeakyReLU(), 'optimizer': 'Ada
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
```

```
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=128, bias=True)
    (1): LeakyReLU()
    (2): Dropout(p=0.07856763066099412, inplace=False)
    (3): Linear(in_features=128, out_features=64, bias=True)
    (4): LeakyReLU()
    (5): Dropout(p=0.07856763066099412, inplace=False)
    (6): Linear(in_features=64, out_features=64, bias=True)
    (7): LeakyReLU()
    (8): Dropout(p=0.07856763066099412, inplace=False)
    (9): Linear(in_features=64, out_features=32, bias=True)
    (10): LeakyReLU()
    (11): Dropout(p=0.07856763066099412, inplace=False)
    (12): Linear(in_features=32, out_features=11, bias=True)
  )
)


       Validate metric              DataLoader 0

          hp_metric               2.362715482711792
           val_acc               0.18021202087402344
          val_loss                2.362715482711792
         valid_mapk              0.24189813435077667


train_model result: {'valid_mapk': 0.24189813435077667, 'val_loss': 2.362715482711792, 'val_a

spotPython tuning: 2.2531075477600098 [#####-----] 50.99%


config: {'l1': 128, 'epochs': 512, 'batch_size': 128, 'act_fn': ELU(), 'optimizer': 'AdamW',
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
```

```
    (0): Linear(in_features=64, out_features=128, bias=True)
    (1): ELU()
    (2): Dropout(p=0.0, inplace=False)
    (3): Linear(in_features=128, out_features=64, bias=True)
    (4): ELU()
    (5): Dropout(p=0.0, inplace=False)
    (6): Linear(in_features=64, out_features=64, bias=True)
    (7): ELU()
    (8): Dropout(p=0.0, inplace=False)
    (9): Linear(in_features=64, out_features=32, bias=True)
    (10): ELU()
    (11): Dropout(p=0.0, inplace=False)
    (12): Linear(in_features=32, out_features=11, bias=True)
  )
)


       Validate metric              DataLoader 0

          hp_metric               2.282184362411499
           val_acc               0.2473498284816742
          val_loss                2.282184362411499
         valid_mapk              0.37861689925193787


train_model result: {'valid_mapk': 0.37861689925193787, 'val_loss': 2.282184362411499, 'val_a

spotPython tuning: 2.2531075477600098 [#####-----] 54.29%


config: {'l1': 128, 'epochs': 64, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax'
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=128, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.0, inplace=False)
```

```
    (3): Linear(in_features=128, out_features=64, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.0, inplace=False)
    (6): Linear(in_features=64, out_features=64, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.0, inplace=False)
    (9): Linear(in_features=64, out_features=32, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.0, inplace=False)
    (12): Linear(in_features=32, out_features=11, bias=True)
  )
)


       Validate metric              DataLoader 0

          hp_metric              2.2808187007904053
           val_acc               0.25088340044021606
          val_loss               2.2808187007904053
         valid_mapk              0.3303273022174835


train_model result: {'valid_mapk': 0.3303273022174835, 'val_loss': 2.2808187007904053, 'val_a

spotPython tuning: 2.2531075477600098 [######----] 59.92%


config: {'l1': 128, 'epochs': 512, 'batch_size': 64, 'act_fn': Tanh(), 'optimizer': 'Adamax'
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=128, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.04660305622600514, inplace=False)
    (3): Linear(in_features=128, out_features=64, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.04660305622600514, inplace=False)
```

```
    (6): Linear(in_features=64, out_features=64, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.04660305622600514, inplace=False)
    (9): Linear(in_features=64, out_features=32, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.04660305622600514, inplace=False)
    (12): Linear(in_features=32, out_features=11, bias=True)
  )
)


       Validate metric              DataLoader 0

          hp_metric              2.2849321365356445
           val_acc               0.24028268456459045
          val_loss               2.2849321365356445
         valid_mapk              0.34888115525245667


train_model result: {'valid_mapk': 0.34888115525245667, 'val_loss': 2.2849321365356445, 'val_

spotPython tuning: 2.2531075477600098 [######----] 62.77%


config: {'l1': 128, 'epochs': 128, 'batch_size': 128, 'act_fn': Tanh(), 'optimizer': 'AdamW'
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=128, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.0, inplace=False)
    (3): Linear(in_features=128, out_features=64, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.0, inplace=False)
    (6): Linear(in_features=64, out_features=64, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.0, inplace=False)
```

```
    (9): Linear(in_features=64, out_features=32, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.0, inplace=False)
    (12): Linear(in_features=32, out_features=11, bias=True)
  )
)


       Validate metric              DataLoader 0

          hp_metric              2.2984087467193604
           val_acc               0.2226148396730423
          val_loss               2.2984087467193604
         valid_mapk               0.322836309671402


train_model result: {'valid_mapk': 0.322836309671402, 'val_loss': 2.2984087467193604, 'val_ac

spotPython tuning: 2.2531075477600098 [#######---] 66.94%


config: {'l1': 64, 'epochs': 512, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax'


_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.1194279848949228, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.1194279848949228, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.1194279848949228, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
```

```
    (10): ReLU()
    (11): Dropout(p=0.1194279848949228, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)


       Validate metric              DataLoader 0

          hp_metric              2.2580628395080566
           val_acc               0.2826855182647705
          val_loss               2.2580628395080566
         valid_mapk              0.37019357085227966


train_model result: {'valid_mapk': 0.37019357085227966, 'val_loss': 2.2580628395080566, 'val_

spotPython tuning: 2.2531075477600098 [#######---] 73.56%


config: {'l1': 64, 'epochs': 512, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'AdamW',


_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.11948497071056166, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.11948497071056166, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.11948497071056166, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): ReLU()
```

```
    (11): Dropout(p=0.11948497071056166, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)


       Validate metric              DataLoader 0

          hp_metric               2.243969202041626
           val_acc               0.28975266218185425
          val_loss                2.243969202041626
         valid_mapk              0.3762538433074951


train_model result: {'valid_mapk': 0.3762538433074951, 'val_loss': 2.243969202041626, 'val_a

spotPython tuning: 2.243969202041626 [########--] 82.22%


config: {'l1': 128, 'epochs': 128, 'batch_size': 256, 'act_fn': ReLU(), 'optimizer': 'NAdam'
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=128, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.13374233520464968, inplace=False)
    (3): Linear(in_features=128, out_features=64, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.13374233520464968, inplace=False)
    (6): Linear(in_features=64, out_features=64, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.13374233520464968, inplace=False)
    (9): Linear(in_features=64, out_features=32, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.13374233520464968, inplace=False)
    (12): Linear(in_features=32, out_features=11, bias=True)
  )
)
```

```
       Validate metric              DataLoader 0

          hp_metric              2.3158271312713623
           val_acc               0.21201413869857788
          val_loss               2.3158271312713623
         valid_mapk              0.32295283675193787


train_model result: {'valid_mapk': 0.32295283675193787, 'val_loss': 2.3158271312713623, 'val_

spotPython tuning: 2.243969202041626 [#########-] 88.90%


config: {'l1': 64, 'epochs': 4096, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.0, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.0, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.0, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.0, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)


       Validate metric              DataLoader 0
```

```
         hp_metric              2.2940661907196045
          val_acc               0.22614841163158417
         val_loss               2.2940661907196045
        valid_mapk              0.3357124328613281


train_model result: {'valid_mapk': 0.3357124328613281, 'val_loss': 2.2940661907196045, 'val_a

spotPython tuning: 2.243969202041626 [##########] 98.67%


config: {'l1': 64, 'epochs': 256, 'batch_size': 64, 'act_fn': LeakyReLU(), 'optimizer': 'Adam
_L_in: 64
_L_out: 11
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): LeakyReLU()
    (2): Dropout(p=0.09881987449847053, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): LeakyReLU()
    (5): Dropout(p=0.09881987449847053, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): LeakyReLU()
    (8): Dropout(p=0.09881987449847053, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): LeakyReLU()
    (11): Dropout(p=0.09881987449847053, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)


       Validate metric              DataLoader 0

         hp_metric              2.2932240962982178
          val_acc               0.2332155406475067
         val_loss               2.2932240962982178
```

```
        valid_mapk              0.34214892983436584


train_model result: {'valid_mapk': 0.34214892983436584, 'val_loss': 2.2932240962982178, 'val_

spotPython tuning: 2.243969202041626 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x10425e1d0>
```

## 17.9 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard.

```
tensorboard --logdir=runs/lightning_logs
```

Further information can be found in the PyTorch Lightning documentation for Tensorboard.

## 17.10 Step 10: Results

After the hyperparameter tuning run is finished, the results can be analyzed as described in Section 12.10.

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

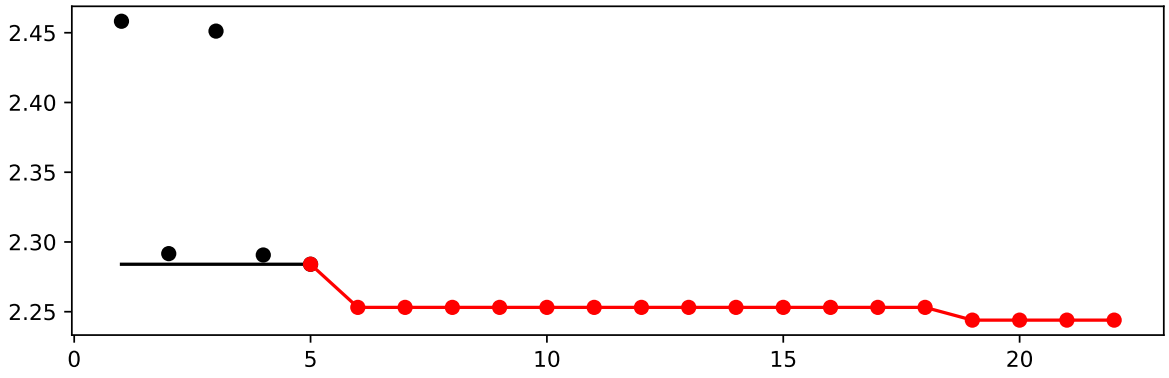



Figure 17.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

```python
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         |     6.0 |    13.0 |                  6.0 | transform_
| epochs         | int    | 4         |     6.0 |    13.0 |                  9.0 | transform_
| batch_size     | int    | 4         |     2.0 |     8.0 |                  7.0 | transform_
| act_fn         | factor | ReLU      |     0.0 |     5.0 |                  2.0 | None
| optimizer      | factor | SGD       |     0.0 |     3.0 |                  1.0 | None
| dropout_prob   | float  | 0.01      |     0.0 |    0.25 | 0.11948497071056166 | None
| lr_mult        | float  | 1.0       |     0.1 |    10.0 |   5.243724341824796 | None
| patience       | int    | 2         |     2.0 |     6.0 |                  5.0 | transform_
| initialization | factor | Default   |     0.0 |     2.0 |                  2.0 | None
```

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

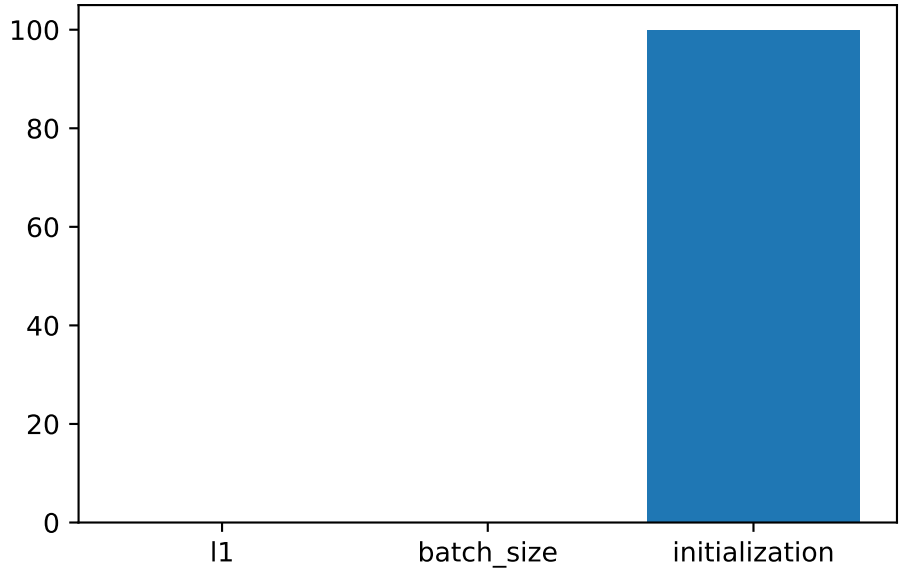



Figure 17.2: Variable importance plot, threshold 0.025.

### 17.10.1 Get the Tuned Architecture

```
from spotPython.light.utils import get_tuned_architecture
config = get_tuned_architecture(spot_tuner, fun_control)
```

- Test on the full data set

```
from spotPython.light.traintest import test_model
test_model(config, fun_control)
```

```
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.11948497071056166, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.11948497071056166, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.11948497071056166, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.11948497071056166, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)
```

| Test metric | DataLoader 0 |
|---|---|
| hp_metric | 2.106801986694336 |
| test_mapk_epoch | 0.484209805727005 |
| val_acc | 0.43422913551330566 |
| val_loss | 2.106801986694336 |

```
test_model result: {'test_mapk_epoch': 0.484209805727005, 'val_loss': 2.106801986694336, 'val
```

```
(2.106801986694336, 0.43422913551330566)
```

```python
from spotPython.light.traintest import load_light_from_checkpoint

model_loaded = load_light_from_checkpoint(config, fun_control)
```

```
Loading model from runs/lightning_logs/64_512_128_ReLU()_AdamW_0.11948497071056166_5.2437243
```

### 17.10.2 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].

```python
from spotPython.light.traintest import cv_model
# set the number of folds to 10
fun_control["k_folds"] = 10
cv_model(config, fun_control)
```

```
k: 0
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.11948497071056166, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.11948497071056166, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.11948497071056166, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.11948497071056166, inplace=False)
```

```
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)
Train Dataset Size: 636
Val Dataset Size: 71


       Validate metric              DataLoader 0

          hp_metric               2.182650089263916
           val_acc               0.35211268067359924
          val_loss                2.182650089263916
         valid_mapk              0.44131454825401306


train_model result: {'valid_mapk': 0.44131454825401306, 'val_loss': 2.182650089263916, 'val_a
k: 1
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.11948497071056166, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.11948497071056166, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.11948497071056166, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.11948497071056166, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)
Train Dataset Size: 636
Val Dataset Size: 71


       Validate metric              DataLoader 0
```

```
         hp_metric              2.1626229286193848
          val_acc               0.3661971688270569
         val_loss               2.1626229286193848
        valid_mapk              0.48122066259384155


train_model result: {'valid_mapk': 0.48122066259384155, 'val_loss': 2.1626229286193848, 'val_
k: 2
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.11948497071056166, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.11948497071056166, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.11948497071056166, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.11948497071056166, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)
Train Dataset Size: 636
Val Dataset Size: 71


       Validate metric              DataLoader 0

         hp_metric              2.3031556606292725
          val_acc               0.23943662643432617
         val_loss               2.3031556606292725
        valid_mapk              0.31220656633377075


train_model result: {'valid_mapk': 0.31220656633377075, 'val_loss': 2.3031556606292725, 'val_
```

```
k: 3
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.11948497071056166, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.11948497071056166, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.11948497071056166, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.11948497071056166, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)
Train Dataset Size: 636
Val Dataset Size: 71


       Validate metric              DataLoader 0

          hp_metric              2.2659246921539307
           val_acc               0.28169015049934387
          val_loss               2.2659246921539307
         valid_mapk              0.3779342770576477


train_model result: {'valid_mapk': 0.3779342770576477, 'val_loss': 2.2659246921539307, 'val_a
k: 4
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): ReLU()
```

```
    (2): Dropout(p=0.11948497071056166, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.11948497071056166, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.11948497071056166, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.11948497071056166, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)
Train Dataset Size: 636
Val Dataset Size: 71


       Validate metric              DataLoader 0

          hp_metric              2.3391501903533936
           val_acc               0.19718310236930847
          val_loss               2.3391501903533936
         valid_mapk              0.2887323796749115


train_model result: {'valid_mapk': 0.2887323796749115, 'val_loss': 2.3391501903533936, 'val_a
k: 5
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.11948497071056166, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.11948497071056166, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.11948497071056166, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
```

```
    (10): ReLU()
    (11): Dropout(p=0.11948497071056166, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)
Train Dataset Size: 636
Val Dataset Size: 71


       Validate metric              DataLoader 0

          hp_metric              2.2540292739868164
           val_acc               0.28169015049934387
          val_loss               2.2540292739868164
         valid_mapk              0.40375587344169617


train_model result: {'valid_mapk': 0.40375587344169617, 'val_loss': 2.2540292739868164, 'val_
k: 6
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.11948497071056166, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.11948497071056166, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.11948497071056166, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.11948497071056166, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)
Train Dataset Size: 636
Val Dataset Size: 71
```

```
       Validate metric              DataLoader 0

          hp_metric              2.3276946544647217
           val_acc               0.2112676054239273
          val_loss               2.3276946544647217
         valid_mapk              0.3004694879055023


train_model result: {'valid_mapk': 0.3004694879055023, 'val_loss': 2.3276946544647217, 'val_a
k: 7
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.11948497071056166, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.11948497071056166, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.11948497071056166, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.11948497071056166, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)
Train Dataset Size: 637
Val Dataset Size: 70


       Validate metric              DataLoader 0

          hp_metric               2.214831590652466
           val_acc               0.3285714387893677
          val_loss                2.214831590652466
         valid_mapk              0.42619046568870544
```

```
train_model result: {'valid_mapk': 0.42619046568870544, 'val_loss': 2.214831590652466, 'val_a
k: 8
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.11948497071056166, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.11948497071056166, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.11948497071056166, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.11948497071056166, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)
Train Dataset Size: 637
Val Dataset Size: 70


       Validate metric              DataLoader 0

          hp_metric               2.279385805130005
           val_acc               0.2571428716182709
          val_loss                2.279385805130005
         valid_mapk              0.3619047701358795


train_model result: {'valid_mapk': 0.3619047701358795, 'val_loss': 2.279385805130005, 'val_ac
k: 9
model: NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
```

```
    (1): ReLU()
    (2): Dropout(p=0.11948497071056166, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.11948497071056166, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.11948497071056166, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.11948497071056166, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
)
Train Dataset Size: 637
Val Dataset Size: 70
```

| Validate metric | DataLoader 0 |
|---|---|
| hp_metric | 2.1907458305358887 |
| val_acc | 0.3571428656578064 |
| val_loss | 2.1907458305358887 |
| valid_mapk | 0.4404761791229248 |

```
train_model result: {'valid_mapk': 0.4404761791229248, 'val_loss': 2.1907458305358887, 'val_a
cv_model mapk result: 0.3834205210208893
```

```
0.3834205210208893
```

> **Note: Evaluation for the Final Comaprison**
>
> - This is the evaluation that will be used in the comparison.

### 17.10.3 Detailed Hyperparameter Plots

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

```
l1:  0.10647080220385724
batch_size:  0.10076313709543509
initialization:  100.0
```

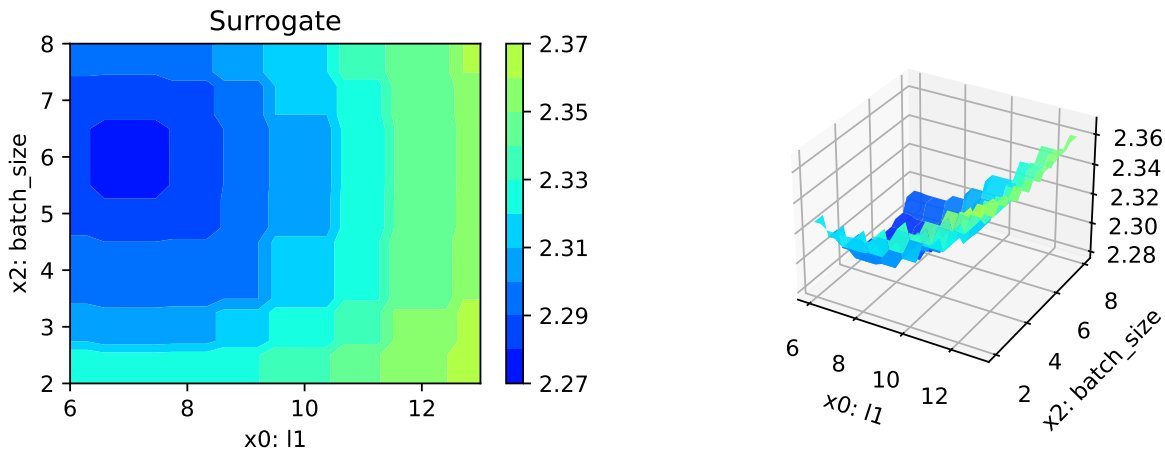
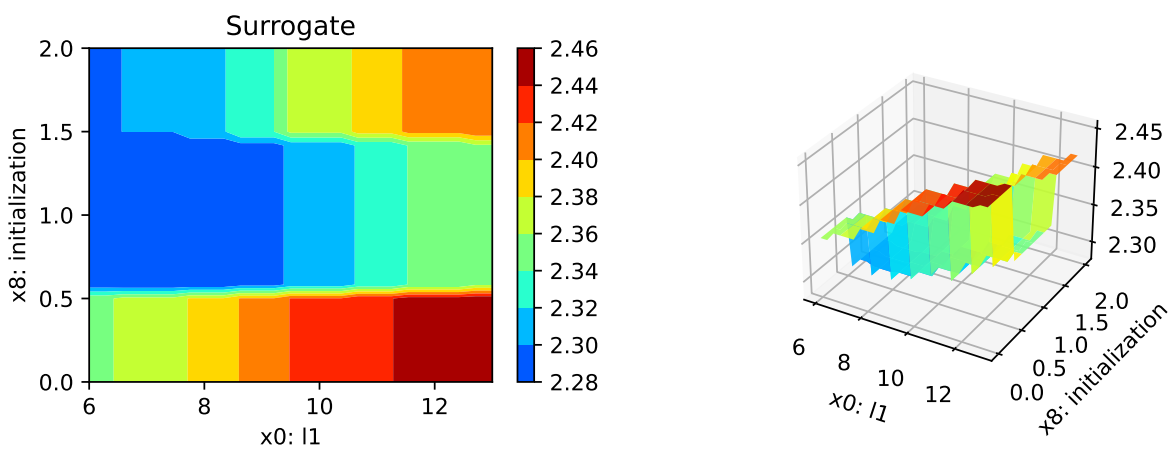
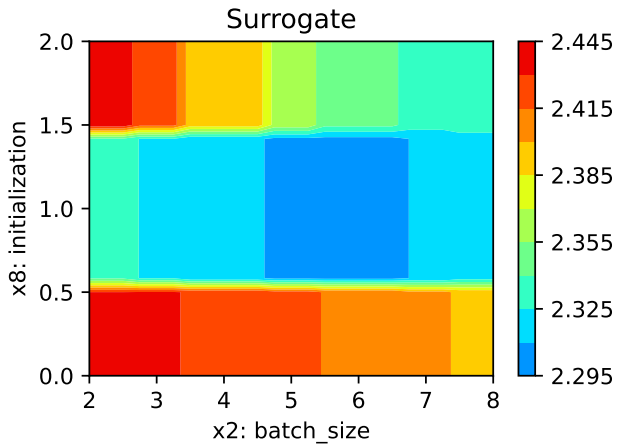
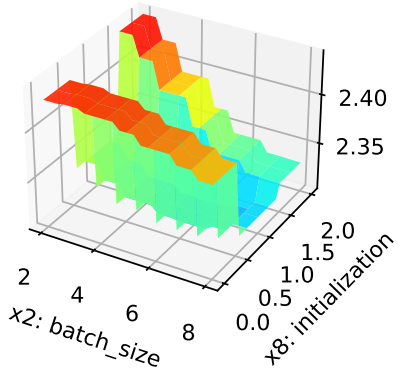



Figure 17.3: Contour plots.






### 17.10.4 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
```

### 17.10.5 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)
```

### 17.10.6 Visualizing the Activation Distribution

> Reference:
>
> - The following code is based on [PyTorch Lightning TUTORIAL 2: ACTIVATION FUNCTIONS], Author: Phillip Lippe, License: [CC BY-SA], Generated: 2023-03-15T09:52:39.179933.

After we have trained the models, we can look at the actual activation values that find inside the model. For instance, how many neurons are set to zero in ReLU? Where do we find most values in Tanh? To answer these questions, we can write a simple function which takes a trained model, applies it to a batch of images, and plots the histogram of the activations inside the network:

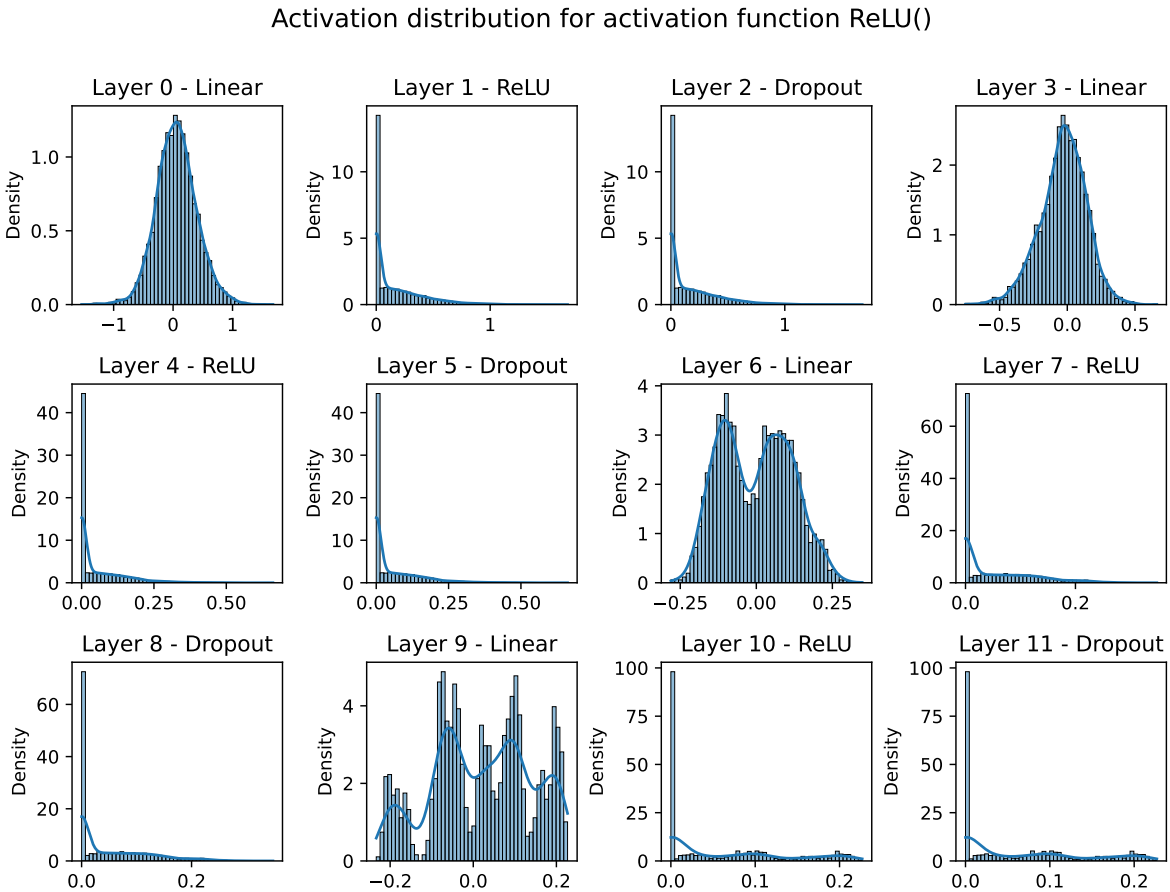
```
from spotPython.torch.activation import Sigmoid, Tanh, ReLU, LeakyReLU, ELU, Swish
act_fn_by_name = {"sigmoid": Sigmoid, "tanh": Tanh, "relu": ReLU, "leakyrelu": LeakyReLU,
```

```
from spotPython.hyperparameters.values import get_one_config_from_X
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
config = get_one_config_from_X(X, fun_control)
model = fun_control["core_model"](**config, _L_in=64, _L_out=11)
model
```

```
NetLightBase(
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (layers): Sequential(
    (0): Linear(in_features=64, out_features=64, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.11948497071056166, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.11948497071056166, inplace=False)
    (6): Linear(in_features=32, out_features=32, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.11948497071056166, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.11948497071056166, inplace=False)
    (12): Linear(in_features=16, out_features=11, bias=True)
  )
```

)

```
from spotPython.utils.eda import visualize_activations
visualize_activations(model, color=f"C{0}")
```




## 17.11 Submission

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import OrdinalEncoder
```

```
import pandas as pd
from sklearn.preprocessing import OrdinalEncoder
```

```
train_df = pd.read_csv('./data/VBDP/train.csv', index_col=0)
# remove the id column
# train_df = train_df.drop(columns=['id'])
n_samples = train_df.shape[0]
n_features = train_df.shape[1] - 1
target_column = "prognosis"
# Encode our prognosis labels as integers for easier decoding later
enc = OrdinalEncoder()
y = enc.fit_transform(train_df[[target_column]])
test_df = pd.read_csv('./data/VBDP/test.csv', index_col=0)
test_df
```

| id | sudden_fever | headache | mouth_bleed | nose_bleed | muscle_pain | joint_pain | vomiting | rash |
|---|---|---|---|---|---|---|---|---|
| 707 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 708 | 1.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 | 1.0 | 1.0 |
| 709 | 1.0 | 1.0 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.0 |
| 710 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 |
| 711 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 1.0 | 0.0 | 0.0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1005 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 1006 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 | 1.0 | 0.0 | 1.0 |
| 1007 | 1.0 | 0.0 | 0.0 | 1.0 | 1.0 | 0.0 | 1.0 | 1.0 |
| 1008 | 1.0 | 0.0 | 1.0 | 1.0 | 1.0 | 0.0 | 1.0 | 0.0 |
| 1009 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 |

```
X_tensor = torch.Tensor(test_df.values)
X_tensor = X_tensor.to(fun_control["device"])
```

```
y = model_loaded(X_tensor)
y.shape
```

```
torch.Size([303, 11])
```

```
# convert the predictions to a numpy array
y = y.cpu().detach().numpy()
y
```

```
array([[3.27358977e-03, 2.57051643e-03, 6.40347078e-02, ...,
        9.00962914e-05, 4.94795650e-01, 1.43416151e-01],
       [9.99702394e-01, 1.74050547e-05, 2.76569256e-09, ...,
        3.05349163e-10, 2.46613922e-07, 2.16867455e-04],
       [6.69627127e-08, 2.49548037e-07, 9.96837735e-01, ...,
        1.26359670e-03, 9.87197654e-06, 1.67813047e-03],
       ...,
       [5.40038968e-07, 1.06684438e-05, 9.82089987e-05, ...,
        9.98893440e-01, 3.04944493e-04, 6.65449479e-04],
       [6.60813898e-02, 2.49378402e-02, 4.98699676e-03, ...,
        6.56329701e-03, 1.38406735e-02, 8.42397034e-01],
       [1.39080284e-05, 4.53270004e-05, 1.03850313e-03, ...,
        9.97359216e-01, 1.12880829e-04, 6.51526498e-04]], dtype=float32)
```

```python
test_sorted_prediction_ids = np.argsort(-y, axis=1)
test_top_3_prediction_ids = test_sorted_prediction_ids[:,:3]
original_shape = test_top_3_prediction_ids.shape
test_top_3_prediction = enc.inverse_transform(test_top_3_prediction_ids.reshape(-1, 1))
test_top_3_prediction = test_top_3_prediction.reshape(original_shape)
test_df['prognosis'] = np.apply_along_axis(lambda x: np.array(' '.join(x), dtype="object")
test_df['prognosis'].reset_index().to_csv('./data/VBDP/submission.csv', index=False)
```

## 17.12 Appendix

### 17.12.1 Differences to the spotPython Approaches for `torch`, `sklearn` and `river`

> **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.

## 17.13 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 12.4. 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.

### 17.13.1 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., 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)}")
```

```python
    print("---------------")
    print(f"Inputs: {inputs}")
    print(f"Targets: {targets}")
    break
```

```
Batch Size: 3
---------------
Inputs: tensor([[0., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 1., 0., 1., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 1., 1., 0., 0., 1., 1., 0., 1., 1., 0., 1., 1.,
         0., 1., 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.],
        [1., 1., 1., 0., 1., 0., 1., 1., 1., 1., 1., 1., 1., 0., 0., 1., 1., 0.,
         1., 1., 1., 1., 1., 0., 1., 0., 1., 1., 0., 0., 0., 1., 1., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 1., 1., 0., 1., 1., 1., 1., 0., 0., 1., 1., 1.,
         1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [1., 1., 0., 0., 0., 1., 0., 1., 1., 1., 1., 0., 1., 0., 1., 1., 1., 0.,
         1., 1., 0., 0., 0., 1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 1., 0., 0.,
         1., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])
Targets: tensor([10,  9,  8])
```

# 18 Documentation of the Sequential Parameter Optimization

This document describes the `Spot` features.

## 18.1 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
```

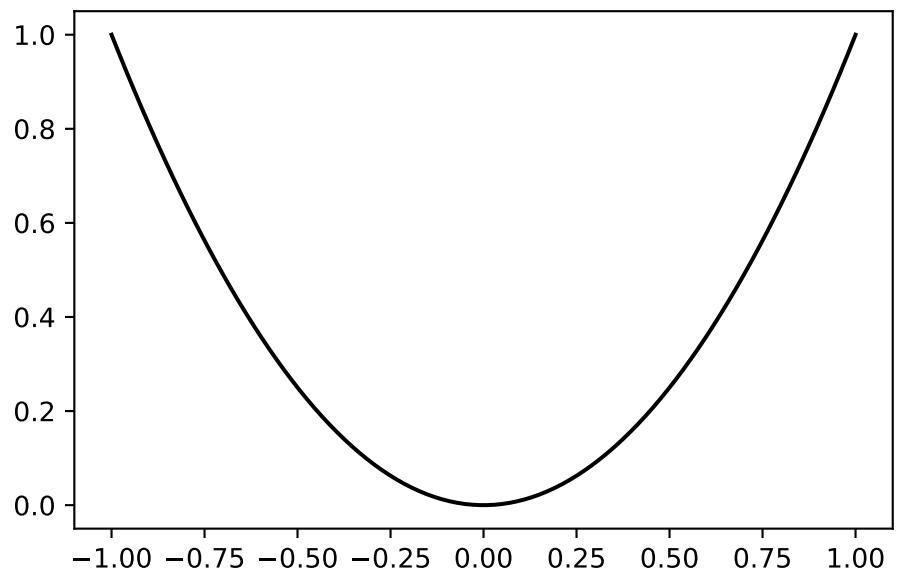
### 18.1.1 The Objective Function

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`.

### 18.1.2 External Parameters

| external parameter | type | description | default | mandatory |
|---|---|---|---|---|
| `fun` | object | objective function | | yes |
| `lower` | array | lower bound | | yes |
| `upper` | array | upper bound | | yes |
| `fun_evals` | int | number of function evaluations | 15 | no |
| `fun_evals` | int | number of function evaluations | 15 | no |
| `fun_control` | dict | noise etc. | {} | n |
| `max_time` | int | max run time budget | `inf` | no |
| `noise` | bool | if repeated evaluations of `fun` results in different values, then `noise` should be set to `True`. | `False` | no |

| external parameter | type | description | default | mandatory |
|---|---|---|---|---|
| `tolerance_x` | float | tolerance for new x solutions. Minimum distance of new solutions, generated by `suggest_new_X`, to already existing solutions. If zero (which is the default), every new solution is accepted. | `0` | no |
| `var_type` | list | list of type information, can be either `"num"` or `"factor"` | `["num"]` | no |
| `infill_criterion` | string | Can be `"y"`, `"s"`, `"ei"` (negative expected improvement), or `"all"` | `"y"` | no |
| `n_points` | int | number of infill points | 1 | no |
| `seed` | int | initial seed. If `Spot.run()` is called twice, different results will be generated. To reproduce results, the `seed` 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 |
|---|---|---|---|---|
| `show_models` | bool | Plot model. Currently only 1-dim functions are supported | `False` | no |
| `design` | object | experimental design | `None` | no |
| `design_control` | dict | control parameters | see below | no |
| `surrogate` | | surrogate model | `kriging` | no |
| `surrogate_control` | dict | control parameters | see below | no |
| `optimizer` | object | optimizer | see below | no |
| `optimizer_control` | 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`

## 18.2 The `fun_control` Dictionary

| external parameter | type | description | default | mandatory |
|---|---|---|---|---|
| `sigma` | float | noise: standard deviation | `0` | yes |
| `seed` | int | seed for rng | `124` | yes |

## 18.3 The `design_control` Dictionary

| external parameter | type | description | default | mandatory |
|---|---|---|---|---|
| `init_size` | int | initial sample size | `10` | yes |

| external parameter | type | description | default | mandatory |
|---|---|---|---|---|
| `repeats` | int | number of repeats of the initial sammples | `1` | yes |

## 18.4 The `surrogate_control` Dictionary

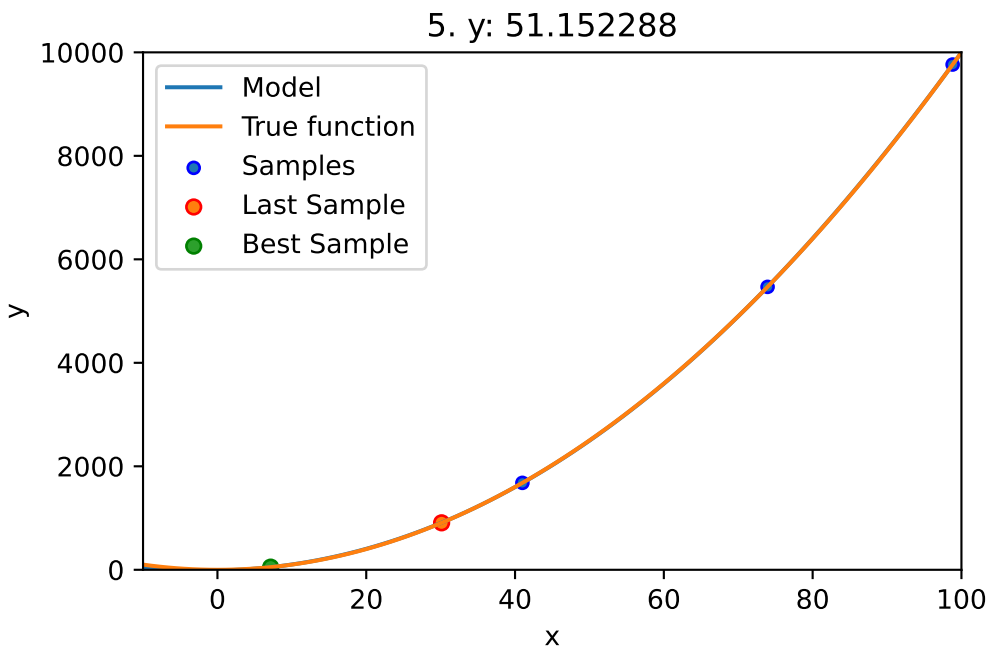
| external parameter | type | description | default | mandatory |
|---|---|---|---|---|
| `noise` | | | | |
| `model_optimizer` | object | optimizer | `differential_evolution` | no |
| `model_fun_evals` | | | | |
| `min_theta` | | | `-3.` | |
| `max_theta` | | | `3.` | |
| `n_theta` | | | `1` | |
| `n_p` | | | `1` | |
| `optim_p` | | | `False` | |
| `cod_type` | | | `"norm"` | |
| `var_type` | | | | |
| `use_cod_y` | bool | | `False` | |

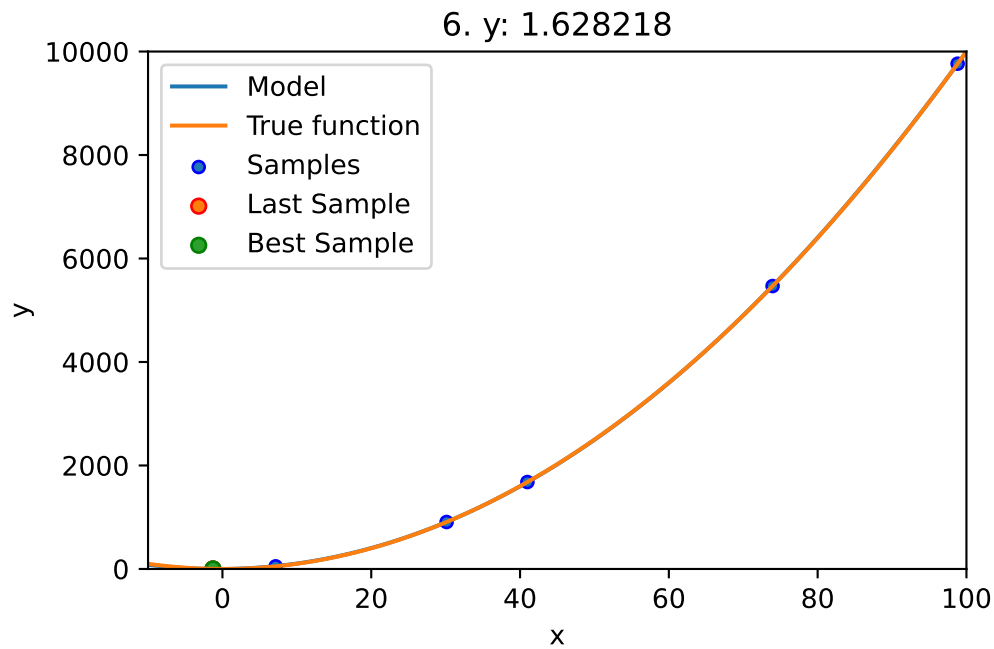
## 18.5 The `optimizer_control` Dictionary

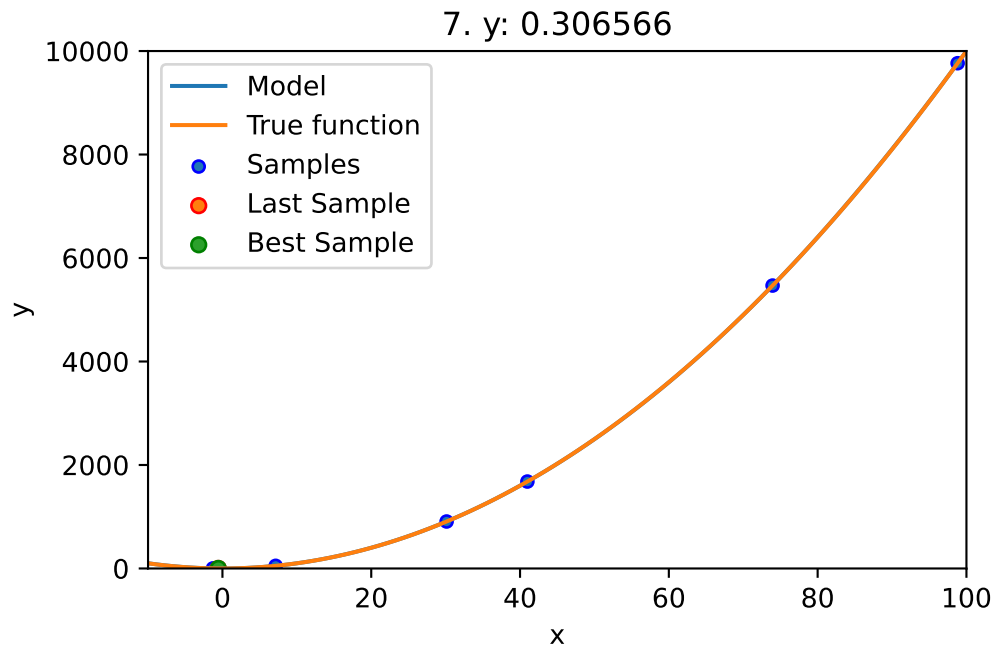
| external parameter | type | description | default | mandatory |
|---|---|---|---|---|
| `max_iter` | int | max number of iterations. Note: these are the cheap evaluations on the surrogate. | `1000` | no |

## 18.6 Run

```
spot_1.run()
```






```
<spotPython.spot.spot.Spot at 0x15208aad0>
```

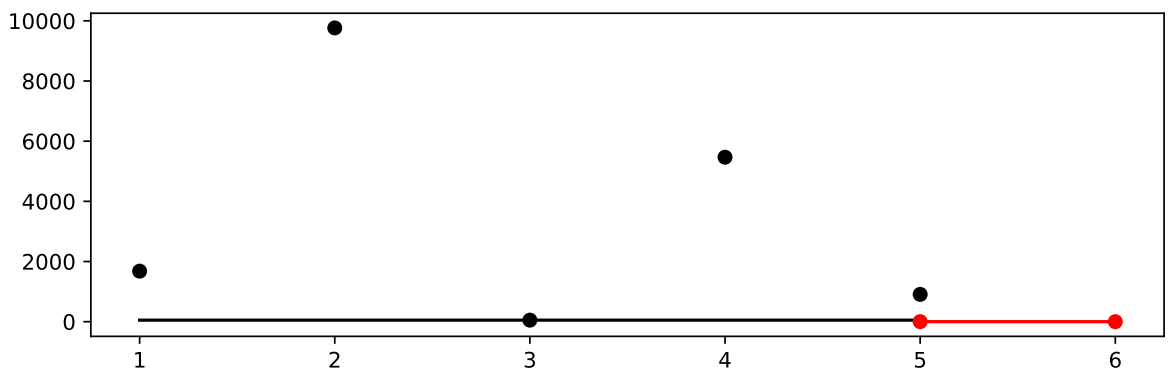
## 18.7 Print the Results

```
spot_1.print_results()
```

```
min y: 0.30656551286610595
x0: -0.5536835855126157

[['x0', -0.5536835855126157]]
```

## 18.8 Show the Progress

```
spot_1.plot_progress()
```

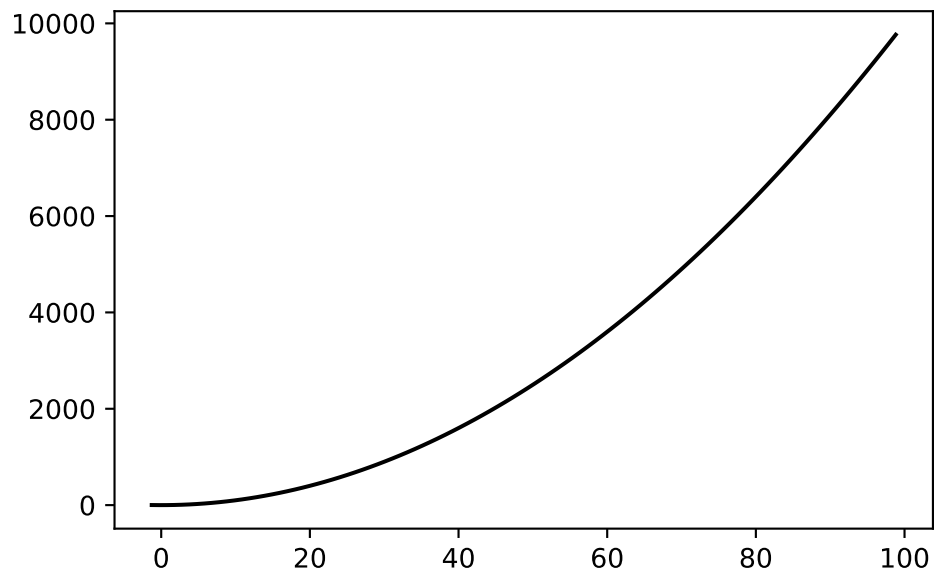



## 18.9 Visualize the Surrogate

- 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>
```




## 18.10 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]
```

## 18.11 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 ]]))
```

## 18.12 Surrogates

### 18.12.1 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 $f$.

## 18.13 Demo/Test: Objective Function Fails

SPOT expects `np.nan` values from failed objective function values. These are handled. Note: SPOT's counter considers only successful executions of the objective function.

```
import numpy as np
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
import numpy as np
from math import inf
# number of initial points:
ni = 20
# number of points
n = 30
```

```
fun = analytical().fun_random_error
lower = np.array([-1])
upper = np.array([1])
design_control={"init_size": ni}

spot_1 = spot.Spot(fun=fun,
            lower = lower,
            upper= upper,
            fun_evals = n,
            show_progress=False,
            design_control=design_control,)
spot_1.run()
# To check whether the run was successfully completed,
# we compare the number of evaluated points to the specified
# number of points.
assert spot_1.y.shape[0] == n
```

```
[ 0.53176481 -0.9053821  -0.02203599 -0.21843718  0.78240941 -0.58120945
 -0.3923345   0.67234256         nan         nan -0.75129705  0.97550354
  0.41757584  0.0786237   0.82585329  0.23700598 -0.49274073 -0.82319082
 -0.17991251  0.1481835 ]
[-1.]

[0.95541987]



[0.17335968]
[-0.58552368]

[-0.20126111]
[-0.60100809]

[-0.97897336]
[-0.2748985]

[0.8359486]
[0.99035591]

[0.01641232]
[0.5629346]
```

## 18.14 PyTorch: Detailed Description of the Data Splitting

### 18.14.1 Description of the "train_hold_out" Setting

The "train_hold_out" setting is used by default. It uses the loss function specfied 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
```

#### 18.14.1.1 Description of the `"test_hold_out"` Setting

It uses the loss function specfied in `fun_control` and the metric specified in `fun_control`.

1. First, the method `HyperTorch().fun_torch` is called.
2. `fun_torc()` 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.

#### 18.14.1.2 Detailed Description of the `"train_cv"` Setting

It uses the loss function specfied in `fun_control` and the metric specified in `fun_control`.

1. First, the method `HyperTorch().fun_torch` is called.
2. `fun_torc()` 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`.

#### 18.14.1.3 Detailed Description of the "test_cv" Setting

It uses the loss function specfied in `fun_control` and the metric specified in `fun_control`.

1. First, the method `HyperTorch().fun_torch` is called.
2. `fun_torc()` 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.

#### 18.14.1.4 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.