Hyperparameter Tuning Cookbook

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

Thomas Bartz-Beielstein

Jun 18, 2023

Table of contents

Preface **3**
Citation . . . 3

1 Introduction: Hyperparameter Tuning **5**
1.1 The Hyperparameter Tuning Software SPOT . . . 6
1.2 Spot as an Optimizer . . . 7
1.3 Example: `Spot` and the Sphere Function . . . 8
1.3.1 The Objective Function: Sphere . . . 8
1.4 `Spot` Parameters: `fun_evals`, `init_size` and `show_models` . . . 10
1.5 Print the Results . . . 12
1.6 Show the Progress . . . 12

2 Multi-dimensional Functions **14**
2.1 Example: `Spot` and the 3-dim Sphere Function . . . 14
2.1.1 The Objective Function: 3-dim Sphere . . . 14
2.1.2 Results . . . 15
2.1.3 A Contour Plot . . . 16
2.2 Conclusion . . . 18
2.3 Exercises . . . 18
2.3.1 The Three Dimensional `fun_cubed` . . . 18
2.3.2 The Ten Dimensional `fun_wing_wt` . . . 19
2.3.3 The Three Dimensional `fun_runge` . . . 19
2.3.4 The Three Dimensional `fun_linear` . . . 19

3 Isotropic and Anisotropic Kriging **20**
3.1 Example: Isotropic `Spot` Surrogate and the 2-dim Sphere Function . . . 20
3.1.1 The Objective Function: 2-dim Sphere . . . 20
3.1.2 Results . . . 21
3.2 Example With Anisotropic Kriging . . . 21
3.2.1 Taking a Look at the `theta` Values . . . 22

4 Exercises **24**
4.1 `fun_branin` . . . 24
4.2 `fun_sin_cos` . . . 24
4.3 `fun_runge` . . . 24
4.4 `fun_wingwt` . . . 25

5 Using sklearn Surrogates in spotPython **26**
5.1 Example: Branin Function with spotPython's Internal Kriging Surrogate . . . 26
5.1.1 The Objective Function Branin . . . 26
5.1.2 Running the surrogate model based optimizer Spot: . . . 27
5.1.3 Print the Results . . . 27
5.1.4 Show the Progress and the Surrogate . . . 27
5.2 Example: Using Surrogates From scikit-learn . . . 28
5.2.1 GaussianProcessRegressor as a Surrogate . . . 29

6 Example: One-dimensional Sphere Function With spotPython's Kriging **31**
6.0.1 Results . . . 36
6.1 Example: Sklearn Model GaussianProcess . . . 37

7 Exercises **44**
7.0.1 DecisionTreeRegressor . . . 44
7.0.2 RandomForestRegressor . . . 44
7.0.3 linear_model.LinearRegression . . . 44
7.0.4 linear_model.Ridge . . . 44
7.1 Exercise 2 . . . 45

8 Sequential Parameter Optimization: Using scipy Optimizers **46**
8.1 The Objective Function Branin . . . 46
8.2 The Optimizer . . . 47
8.3 Print the Results . . . 48
8.4 Show the Progress . . . 48

9 Exercises **50**
9.1 dual_annealing . . . 50
9.2 direct . . . 50
9.3 shgo . . . 50
9.4 basinhopping . . . 50
9.5 Performance Comparison . . . 50

10 Sequential Parameter Optimization: Gaussian Process Models **52**
10.1 Gaussian Processes Regression: Basic Introductory scikit-learn Example . . 52
10.1.1 Train and Test Data . . . 53
10.1.2 Building the Surrogate With Sklearn . . . 53
10.1.3 Plotting the SklearnModel . . . 53
10.1.4 The spotPython Version . . . 54
10.1.5 Visualizing the Differences Between the spotPython and the sklearn Model Fits . . . 55

11 Exercises **57**
11.1 Schonlau Example Function . . . 57

11.2 `Forrester Example Function` . . . 57
11.3 `fun_runge Function (1-dim)` . . . 58
11.4 `fun_cubed (1-dim)` . . . 58
11.5 The Effect of Noise . . . 59

12 Expected Improvement **60**
12.1 Example: `Spot` and the 1-dim Sphere Function . . . 60
12.1.1 The Objective Function: 1-dim Sphere . . . 60
12.1.2 Results . . . 61
12.2 Same, but with EI as infill_criterion . . . 61
12.3 Non-isotropic Kriging . . . 62
12.4 Using `sklearn` Surrogates . . . 64
12.4.1 The spot Loop . . . 64
12.4.2 spot: The Initial Model . . . 66
12.4.3 Init: Build Initial Design . . . 66
12.4.4 Evaluate . . . 69
12.4.5 Build Surrogate . . . 69
12.4.6 A Simple Predictor . . . 69
12.5 Gaussian Processes regression: basic introductory example . . . 69
12.6 The Surrogate: Using scikit-learn models . . . 72
12.7 Additional Examples . . . 74
12.7.1 Optimize on Surrogate . . . 78
12.7.2 Evaluate on Real Objective . . . 78
12.7.3 Impute / Infill new Points . . . 78
12.8 Tests . . . 78
12.9 EI: The Famous Schonlau Example . . . 79
12.10 EI: The Forrester Example . . . 81
12.11 Noise . . . 84
12.12 Cubic Function . . . 87
12.13 Factors . . . 93

13 Hyperparameter Tuning and Noise **95**
13.1 Example: `Spot` and the Noisy Sphere Function . . . 95
13.1.1 The Objective Function: Noisy Sphere . . . 95
13.2 Print the Results . . . 99
13.3 Noise and Surrogates: The Nugget Effect . . . 99
13.3.1 The Noisy Sphere . . . 99
13.4 Exercises . . . 102
13.4.1 Noisy `fun_cubed` . . . 102
13.4.2 `fun_runge` . . . 103
13.4.3 `fun_forrester` . . . 103
13.4.4 `fun_xsin` . . . 103

14 Handling Noise: Optimal Computational Budget Allocation in `Spot` **104**
14.1 Example: `Spot`, OCBA, and the Noisy Sphere Function . . . 104
14.1.1 The Objective Function: Noisy Sphere . . . 104
14.2 Print the Results . . . 114
14.3 Noise and Surrogates: The Nugget Effect . . . 115
14.3.1 The Noisy Sphere . . . 115
14.4 Exercises . . . 118
14.4.1 Noisy `fun_cubed` . . . 118
14.4.2 `fun_runge` . . . 118
14.4.3 `fun_forrester` . . . 118
14.4.4 `fun_xsin` . . . 119

15 HPT: sklearn SVC on Moons Data **120**
15.1 Step 1: Setup . . . 120
15.2 Step 2: Initialization of the Empty `fun_control` Dictionary . . . 121
15.3 Step 3: SKlearn Load Data (Classification) . . . 121
15.4 Step 4: Specification of the Preprocessing Model . . . 123
15.5 Step 5: Select Model (`algorithm`) and `core_model_hyper_dict` . . . 124
15.6 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model` . . . 126
15.6.1 Modify hyperparameter of type numeric and integer (boolean) . . . 127
15.6.2 Modify hyperparameter of type factor . . . 127
15.6.3 Optimizers . . . 127
15.7 Step 7: Selection of the Objective (Loss) Function . . . 128
15.7.1 Predict Classes or Class Probabilities . . . 128
15.8 Step 8: Calling the SPOT Function . . . 128
15.8.1 Preparing the SPOT Call . . . 128
15.8.2 The Objective Function . . . 129
15.8.3 Run the `Spot` Optimizer . . . 129
15.8.4 Starting the Hyperparameter Tuning . . . 130
15.9 Step 9: Results . . . 131
15.9.1 Show variable importance . . . 133
15.9.2 Get Default Hyperparameters . . . 134
15.9.3 Get SPOT Results . . . 134
15.9.4 Plot: Compare Predictions . . . 135
15.9.5 Detailed Hyperparameter Plots . . . 138
15.9.6 Parallel Coordinates Plot . . . 139
15.9.7 Plot all Combinations of Hyperparameters . . . 139

16 HPT: PyTorch With fashionMNIST **140**
16.1 Step 1: Setup . . . 140
16.2 Step 2: Initialization of the Empty `fun_control` Dictionary . . . 142

16.3 Step 3: PyTorch Data Loading . . . 142
16.3.1 Load fashionMNIST Data . . . 142
16.4 Step 4: Specification of the Preprocessing Model . . . 143
16.5 Step 5: Select Model (`algorithm`) and `core_model_hyper_dict` . . . 143
16.5.1 The Search Space . . . 144
16.5.2 Configuring the Search Space With spotPython . . . 144
16.6 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model` . . . 146
16.6.1 Modify hyperparameter of type numeric and integer (boolean) . . . 146
16.6.2 Modify hyperparameter of type factor . . . 147
16.6.3 Optimizers . . . 147
16.7 Step 7: Selection of the Objective (Loss) Function . . . 147
16.7.1 Evaluation . . . 147
16.7.2 Metric . . . 148
16.8 Step 8: Calling the SPOT Function . . . 148
16.8.1 Preparing the SPOT Call . . . 148
16.8.2 The Objective Function `fun_torch` . . . 149
16.8.3 Starting the Hyperparameter Tuning . . . 149
16.9 Step 9: Tensorboard . . . 154
16.10 Step 10: Results . . . 154
16.10.1 Show variable importance . . . 156
16.10.2 Get the Tuned Architecture (SPOT Results) . . . 156
16.10.3 Get Default Hyperparameters . . . 157
16.10.4 Evaluation of the Default and the Tuned Architectures . . . 157
16.10.5 Detailed Hyperparameter Plots . . . 160
16.10.6 Parallel Coordinates Plot . . . 161
16.10.7 Plot all Combinations of Hyperparameters . . . 161

17 HPT: PyTorch With cifar10 Data 162
17.1 Step 1: Setup . . . 162
17.2 Step 2: Initialization of the Empty `fun_control` Dictionary . . . 164
17.3 Step 3: PyTorch Data Loading . . . 164
17.3.1 Load Data Cifar10 Data . . . 164
17.4 Step 4: Specification of the Preprocessing Model . . . 165
17.5 Step 5: Select Model (`algorithm`) and `core_model_hyper_dict` . . . 165
17.5.1 Implementing a Configurable Neural Network With spotPython . . . 165
17.5.2 The Search Space . . . 166
17.5.3 Configuring the Search Space With spotPython . . . 166
17.6 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model` . . . 167
17.6.1 Step 5: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model` . . . 168
17.6.2 Modify hyperparameter of type factor . . . 168

17.6.3 Optimizers . . . 168
17.7 Step 7: Selection of the Objective (Loss) Function . . . 169
17.7.1 Evaluation . . . 169
17.7.2 Metric . . . 169
17.8 Step 8: Calling the SPOT Function . . . 170
17.8.1 Preparing the SPOT Call . . . 170
17.8.2 The Objective Function `fun_torch` . . . 170
17.8.3 Starting the Hyperparameter Tuning . . . 171
17.9 Step 9: Tensorboard . . . 175
17.10 Step 10: Results . . . 175
17.10.1 Show variable importance . . . 177
17.10.2 Get the Tuned Architecture (SPOT Results) . . . 177
17.10.3 Evaluation of the Tuned Architecture . . . 178
17.10.4 Cross-validated Evaluations . . . 179
17.10.5 Detailed Hyperparameter Plots . . . 180
17.10.6 Parallel Coordinates Plot . . . 182
17.10.7 Plot all Combinations of Hyperparameters . . . 182

18 HPT: River **183**
18.1 Step 1: Setup . . . 183
18.1.1 `river` Hyperparameter Tuning: HATR with Friedman Drift Data . . . 183
18.2 Step 2: Initialization of the `fun_control` Dictionary . . . 184
18.3 Step 3: Load the Friedman Drift Data . . . 184
18.4 Step 4: Specification of the Preprocessing Model . . . 185
18.5 Step 5: Select `algorithm` and `core_model_hyper_dict` . . . 186
18.6 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model` . . . 189
18.6.1 Modify hyperparameter of type factor . . . 189
18.6.2 Modify hyperparameter of type numeric and integer (boolean) . . . 189
18.7 Step 7: Selection of the Objective (Loss) Function . . . 189
18.8 Step 8: Calling the SPOT Function . . . 190
18.8.1 Prepare the SPOT Parameters . . . 190
18.8.2 Run the `Spot` Optimizer . . . 191
18.9 Step 9: Results . . . 192
18.9.1 Show variable importance . . . 194
18.9.2 Build and Evaluate HTR Model with Tuned Hyperparameters . . . 195
18.9.3 The Large Data Set (k=0.2) . . . 195
18.9.4 Get Default Hyperparameters . . . 196
18.9.5 Get SPOT Results . . . 199
18.9.6 Visualize Regression Trees . . . 203
18.9.7 Spot Model . . . 203
18.9.8 Detailed Hyperparameter Plots . . . 205
18.9.9 Parallel Coordinates Plots . . . 206

18.9.10 Plot all Combinations of Hyperparameters . . . 206

19 HPT: PyTorch With `spotPython` and Ray Tune on CIFAR10 207
19.1 Step 1: Setup . . . 208
19.2 Step 2: Initialization of the `fun_control` Dictionary . . . 209
19.3 Step 3: PyTorch Data Loading . . . 209
19.4 Step 4: Specification of the Preprocessing Model . . . 210
19.5 Step 5: Select Model (`algorithm`) and `core_model_hyper_dict` . . . 211
19.5.1 The `Net_Core` class . . . 213
19.5.2 Comparison of the Approach Described in the PyTorch Tutorial With spotPython . . . 213
19.5.3 The Search Space: Hyperparameters . . . 214
19.5.4 Configuring the Search Space With Ray Tune . . . 214
19.5.5 Configuring the Search Space With spotPython . . . 215
19.6 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model` . . . 217
19.6.1 Optimizers . . . 218
19.7 Step 7: Selection of the Objective (Loss) Function . . . 220
19.7.1 Evaluation: Data Splitting . . . 220
19.7.2 Hold-out Data Split . . . 220
19.7.3 Cross-Validation . . . 221
19.7.4 Overview of the Evaluation Settings . . . 221
19.7.5 Evaluation: Loss Functions and Metrics . . . 223
19.8 Step 8: Calling the SPOT Function . . . 224
19.8.1 Preparing the SPOT Call . . . 224
19.8.2 The Objective Function `fun_torch` . . . 225
19.8.3 Using Default Hyperparameters or Results from Previous Runs . . . 225
19.8.4 Starting the Hyperparameter Tuning . . . 225
19.9 Step 9: Tensorboard . . . 235
19.9.1 Tensorboard: Start Tensorboard . . . 235
19.9.2 Saving the State of the Notebook . . . 235
19.10 Step 10: Results . . . 237
19.10.1 Get the Tuned Architecture (SPOT Results) . . . 239
19.10.2 Get Default Hyperparameters . . . 240
19.10.3 Evaluation of the Default Architecture . . . 240
19.10.4 Evaluation of the Tuned Architecture . . . 242
19.10.5 Detailed Hyperparameter Plots . . . 243
19.11 Summary and Outlook . . . 244
19.12 Appendix . . . 245
19.12.1 Sample Output From Ray Tune's Run . . . 245

20 HPT: sklearn RandomForestClassifier VBDP Data 247
20.1 Step 1: Setup . . . 247

20.2 Step 2: Initialization of the Empty `fun_control` Dictionary 248
20.3 Step 3: PyTorch Data Loading 249
20.3.1 Load Data: Classification VBDP 249
20.3.2 Holdout Train and Test Data 249
20.4 Step 4: Specification of the Preprocessing Model 250
20.5 Step 5: Select Model (`algorithm`) and `core_model_hyper_dict` 251
20.6 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model` 253
20.6.1 Modify hyperparameter of type numeric and integer (boolean) 253
20.6.2 Modify hyperparameter of type factor 253
20.6.3 Optimizers 254
20.6.4 Selection of the Objective: Metric and Loss Functions 254
20.7 Step 7: Selection of the Objective (Loss) Function 254
20.7.1 Metric Function 254
20.7.2 Evaluation on Hold-out Data 255
20.7.3 OOB Score 256
20.8 Step 8: Calling the SPOT Function 257
20.8.1 Preparing the SPOT Call 257
20.8.2 The Objective Function 257
20.8.3 Run the `Spot` Optimizer 258
20.9 Step 9: Tensorboard 260
20.10 Step 10: Results 261
20.10.1 Show variable importance 262
20.10.2 Get Default Hyperparameters 262
20.10.3 Get SPOT Results 263
20.10.4 Evaluate SPOT Results 264
20.10.5 Handling Non-deterministic Results 265
20.10.6 Evalution of the Default Hyperparameters 265
20.10.7 Plot: Compare Predictions 266
20.10.8 Cross-validated Evaluations 268
20.10.9 Detailed Hyperparameter Plots 269
20.10.10 Parallel Coordinates Plot 272
20.10.11 Plot all Combinations of Hyperparameters 272

21 HPT: sklearn XGB Classifier VBDP Data 273
21.1 Step 1: Setup 273
21.2 Step 2: Initialization of the Empty `fun_control` Dictionary 274
21.3 Step 3: PyTorch Data Loading 275
21.3.1 1. Load Data: Classification VBDP 275
21.3.2 Holdout Train and Test Data 275
21.4 Step 4: Specification of the Preprocessing Model 276
21.5 Step 5: Select Model (`algorithm`) and `core_model_hyper_dict` 277

21.6 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model` . . . 279
21.6.1 Modify hyperparameter of type numeric and integer (boolean) . . . 279
21.6.2 Modify hyperparameter of type factor . . . 279
21.6.3 Optimizers . . . 280
21.7 Step 7: Selection of the Objective (Loss) Function . . . 280
21.7.1 Evaluation . . . 280
21.7.2 Selection of the Objective: Metric and Loss Functions . . . 280
21.7.3 Loss Function . . . 280
21.7.4 Metric Function . . . 280
21.7.5 Evaluation on Hold-out Data . . . 282
21.8 Step 8: Calling the SPOT Function . . . 282
21.8.1 Preparing the SPOT Call . . . 282
21.8.2 The Objective Function . . . 283
21.8.3 Run the `Spot` Optimizer . . . 283
21.9 Step 9: Tensorboard . . . 285
21.10 Step 10: Results . . . 285
21.10.1 Show variable importance . . . 287
21.10.2 Get Default Hyperparameters . . . 287
21.10.3 Get SPOT Results . . . 288
21.10.4 Evaluate SPOT Results . . . 289
21.10.5 Handling Non-deterministic Results . . . 290
21.10.6 Evalution of the Default Hyperparameters . . . 290
21.10.7 Plot: Compare Predictions . . . 291
21.10.8 Cross-validated Evaluations . . . 293
21.10.9 Detailed Hyperparameter Plots . . . 294
21.10.10 Parallel Coordinates Plot . . . 304
21.10.11 Plot all Combinations of Hyperparameters . . . 304

22 HPT: sklearn SVC VBDP Data 305
22.1 Step 1: Setup . . . 305
22.2 Step 2: Initialization of the Empty `fun_control` Dictionary . . . 306
22.3 Step 3: PyTorch Data Loading . . . 307
22.3.1 1. Load Data: Classification VBDP . . . 307
22.3.2 Holdout Train and Test Data . . . 307
22.4 Step 4: Specification of the Preprocessing Model . . . 308
22.5 Step 5: Select Model (`algorithm`) and `core_model_hyper_dict` . . . 309
22.6 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model` . . . 311
22.6.1 Modify hyperparameter of type numeric and integer (boolean) . . . 311
22.6.2 Modify hyperparameter of type factor . . . 311
22.6.3 Optimizers . . . 311
22.6.4 Selection of the Objective: Metric and Loss Functions . . . 312

22.7 Step 7: Selection of the Objective (Loss) Function . . . 312
22.7.1 Metric Function . . . 312
22.7.2 Evaluation on Hold-out Data . . . 313
22.8 Step 8: Calling the SPOT Function . . . 314
22.8.1 Preparing the SPOT Call . . . 314
22.8.2 The Objective Function . . . 315
22.8.3 Run the `Spot` Optimizer . . . 315
22.9 Step 9: Tensorboard . . . 319
22.10 Step 10: Results . . . 320
22.10.1 Show variable importance . . . 321
22.10.2 Get Default Hyperparameters . . . 321
22.10.3 Get SPOT Results . . . 322
22.10.4 Evaluate SPOT Results . . . 323
22.10.5 Handling Non-deterministic Results . . . 324
22.10.6 Evalution of the Default Hyperparameters . . . 324
22.10.7 Plot: Compare Predictions . . . 325
22.10.8 Cross-validated Evaluations . . . 327
22.10.9 Detailed Hyperparameter Plots . . . 328
22.10.10 Parallel Coordinates Plot . . . 331
22.10.11 Plot all Combinations of Hyperparameters . . . 331

23 HPT: sklearn KNN Classifier VBDP Data 332
23.1 Step 1: Setup . . . 332
23.2 Step 2: Initialization of the Empty `fun_control` Dictionary . . . 333
23.2.1 Load Data: Classification VBDP . . . 333
23.2.2 Holdout Train and Test Data . . . 334
23.3 Step 4: Specification of the Preprocessing Model . . . 335
23.4 Step 5: Select Model (`algorithm`) and `core_model_hyper_dict` . . . 336
23.5 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model` . . . 337
23.5.1 Modify hyperparameter of type numeric and integer (boolean) . . . 337
23.5.2 Modify hyperparameter of type factor . . . 338
23.5.3 Optimizers . . . 338
23.5.4 Selection of the Objective: Metric and Loss Functions . . . 338
23.6 Step 7: Selection of the Objective (Loss) Function . . . 338
23.6.1 Metric Function . . . 339
23.6.2 Evaluation on Hold-out Data . . . 340
23.7 Step 8: Calling the SPOT Function . . . 340
23.7.1 Preparing the SPOT Call . . . 340
23.7.2 The Objective Function . . . 341
23.7.3 Run the `Spot` Optimizer . . . 341
23.8 Step 9: Tensorboard . . . 345

23.9 Step 10: Results . . . 346
23.9.1 Show variable importance . . . 346
23.9.2 Get Default Hyperparameters . . . 347
23.9.3 Get SPOT Results . . . 348
23.9.4 Evaluate SPOT Results . . . 348
23.9.5 Handling Non-deterministic Results . . . 349
23.9.6 Evalution of the Default Hyperparameters . . . 350
23.9.7 Plot: Compare Predictions . . . 350
23.9.8 Cross-validated Evaluations . . . 352
23.9.9 Detailed Hyperparameter Plots . . . 353
23.9.10 Parallel Coordinates Plot . . . 354
23.9.11 Plot all Combinations of Hyperparameters . . . 354

24 HPT PyTorch: Regression 355
24.1 Step 1: Setup . . . 355
24.2 Step 2: Initialization of the `fun_control` Dictionary . . . 357
24.3 Step 3: PyTorch Data Loading . . . 357
24.4 Step 4: Specification of the Preprocessing Model . . . 359
24.5 Step 5: Select Model (`algorithm`) and `core_model_hyper_dict` . . . 359
24.5.1 Implementing a Configurable Neural Network With spotPython . . . 359
24.5.2 The Search Space . . . 361
24.5.3 Configuring the Search Space With spotPython . . . 361
24.6 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model` . . . 363
24.6.1 Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model` . . . 363
24.6.2 Optimizers . . . 363
24.7 Step 7: Selection of the Objective (Loss) Function . . . 364
24.7.1 Evaluation . . . 364
24.7.2 Loss Functions and Metrics . . . 364
24.7.3 Metric . . . 364
24.8 Step 8: Calling the SPOT Function . . . 364
24.8.1 Preparing the SPOT Call . . . 364
24.8.2 The Objective Function `fun_torch` . . . 365
24.8.3 Starting the Hyperparameter Tuning . . . 365
24.9 Step 9: Tensorboard . . . 433
24.10 Step 10: Results . . . 433
24.10.1 Get the Tuned Architecture (SPOT Results) . . . 434
24.10.2 Evaluation of the Tuned Architecture . . . 435
24.10.3 Cross-validated Evaluations . . . 436
24.10.4 Detailed Hyperparameter Plots . . . 446
24.10.5 Parallel Coordinates Plot . . . 452
24.11 Summary and Outlook . . . 452

25 HPT: PyTorch With VBDP **454**
25.1 Step 1: Setup . . . 455
25.2 Step 2: Initialization of the `fun_control` Dictionary . . . 456
25.3 Step 3: PyTorch Data Loading . . . 456
25.3.1 1. Load VBDP Data . . . 456
25.3.2 Check content of the target column . . . 458
25.4 Step 4: Specification of the Preprocessing Model . . . 459
25.5 Step 5: Select `algorithm` and `core_model_hyper_dict` . . . 459
25.5.1 Implementing a Configurable Neural Network With spotPython . . . 459
25.5.2 Add the NN Model to the fun_control Dictionary . . . 459
25.6 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model` . . . 461
25.6.1 Optimizers . . . 462
25.7 Step 7: Selection of the Objective (Loss) Function . . . 462
25.7.1 Evaluation . . . 462
25.7.2 Loss Functions and Metrics . . . 462
25.7.3 Metric . . . 463
25.8 Step 8: Calling the SPOT Function . . . 463
25.8.1 Preparing the SPOT Call . . . 463
25.8.2 The Objective Function `fun_torch` . . . 464
25.8.3 Starting the Hyperparameter Tuning . . . 465
25.9 Step 9: Tensorboard . . . 470
25.10 Step 10: Results . . . 470
25.10.1 Get the Tuned Architecture . . . 472
25.10.2 Evaluation of the Tuned Architecture . . . 473
25.10.3 Cross-validated Evaluations . . . 474
25.10.4 Detailed Hyperparameter Plots . . . 477
25.10.5 Parallel Coordinates Plot . . . 478
25.10.6 Plot all Combinations of Hyperparameters . . . 478

26 Documentation of the Sequential Parameter Optimization **479**
26.1 Example: spot . . . 479
26.1.1 The Objective Function . . . 479
26.1.2 External Parameters . . . 481
26.2 The `fun_control` Dictionary . . . 484
26.3 The `design_control` Dictionary . . . 484
26.4 The `surrogate_control` Dictionary . . . 485
26.5 The `optimizer_control` Dictionary . . . 485
26.6 Run . . . 486
26.7 Print the Results . . . 488
26.8 Show the Progress . . . 488
26.9 Visualize the Surrogate . . . 488
26.10 Init: Build Initial Design . . . 489

26.11 Replicability 490
26.12 Surrogates 491
26.12.1 A Simple Predictor 491
26.13 Demo/Test: Objective Function Fails 491
26.14 PyTorch: Detailed Description of the Data Splitting 493
26.14.1 Description of the `"train_hold_out"` Setting 493

References **504**

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 19 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 19.1 describes the setup of the tuners. Section 19.3 describes the data loading. Section 19.5 describes the model to be tuned. The search space is introduced in Section 19.5.3. Optimizers are presented in Section 19.6.1. How to split the data in train, validation, and test sets is described in Section 19.7.1. The selection of the loss function and metrics is described in Section 19.7.5. Section 19.8.1 describes the preparation of the `spotPython` call. The objective function is described in Section 19.8.2. How to use results from previous runs and default hyperparameter configurations is described in Section 19.8.3. Starting the tuner is shown in Section 19.8.4. TensorBoard can be used to visualize the results as shown in Section 19.9. Results are discussed and explained in Section 19.10. Finally, Section 19.11 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 19.3.
2. Specification of the preprocessing model, see Section 19.4. 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 **?@sec-selection-of-the-algorithm**. 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 19.6.0.1. 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 19.7.5.
6. Calling SPOT with the corresponding parameters, see Section 19.8.4. The results are stored in a dictionary and are available for further analysis.
7. Presentation, visualization and interpretation of the results, see Section 19.10.

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 0x105546e90>
```

```
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 0x16c1a35b0>
```

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 0x107b83880>
```

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 0x148afb8b0>
```

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 0x1495af640>
```

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()
```

4 Exercises

4.1 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,
```

4.2 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`.

4.3 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`.

4.4 `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`.

5 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
```

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

5.1.1 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
```

5.1.2 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 0x105772530>
```

5.1.3 Print the Results

```
spot_2.print_results()
```

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

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

5.1.4 Show the Progress and the Surrogate

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

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

5.2 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)
```

5.2.1 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 0x295793df0>
```

```
spot_2_GP.plot_progress()
```

```
spot_2_GP.print_results()
```

```
min y: 0.39834138199933733
x0: 3.151313040413957
x1: 2.2668026911984738

[['x0', 3.151313040413957], ['x1', 2.2668026911984738]]
```

6 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 0x297884190>
```

6.0.1 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()
```

6.1 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 0x2a8c048e0>
```

```
spot_1_GP.print_results()
```

```
min y: 5.680275338066567e-09
x0: -7.53676013819371e-05

[['x0', -7.53676013819371e-05]]
```

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

```
spot_1_GP.plot_model()
```

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

7 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.

7.0.1 `DecisionTreeRegressor`

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

7.0.2 `RandomForestRegressor`

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

7.0.3 `linear_model.LinearRegression`

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

7.0.4 `linear_model.Ridge`

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

7.1 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`

8 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
```

8.1 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
```

8.2 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 0x169673a60>
```

8.3 Print the Results

```
spot_de.print_results()
```

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

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

8.4 Show the Progress

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

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

9 Exercises

9.1 `dual_annealing`

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

9.2 `direct`

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

9.3 `shgo`

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

9.4 `basinhopping`

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

9.5 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?

10 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
```

10.1 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.

10.1.1 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]
```

10.1.2 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)
```

10.1.3 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")
```

10.1.4 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")
```

10.1.5 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")
```

Comparing Mean Predictions
$f(x) = x\sin(x)$
Observations
spotPython Mean prediction
Sklearn Mean Prediction
$f(x)$
x
8
6
4
2
0
−2
−4
−6
0
2
4
6
8
10

11 Exercises

11.1 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.

11.2 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}
```

11.3 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}
```

11.4 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}
```

11.5 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?

12 Expected Improvement

12.1 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
```

12.1.1 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 0x163163880>
```

12.1.2 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)
```

12.2 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 0x16885d060>
```

```
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]]
```

12.3 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 0x168ac0b50>
```

```
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()
```

12.4 Using `sklearn` Surrogates

12.4.1 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 12.1: 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.

12.4.2 spot: The Initial Model

12.4.2.1 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 0x16abc6da0>
```

```
spot_ei.plot_progress()
```

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

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

12.4.3 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 ]]))
```

12.4.4 Evaluate

12.4.5 Build Surrogate

12.4.6 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.

12.5 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")
```

12.6 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 0x1633ba0b0>
```

```
spot_GP.y
```

```
array([ 69.32459936, 152.38491454, 107.92560483,  24.51465459,
        76.73500031, 128.1584069 ,   2.18817888,  75.03822603,
         2.19361268,   1.18420699,  24.2718825 ,   1.43444843,
         0.73219702,  14.30411207,  17.17852435])
```

```
spot_GP.plot_progress()
```

```
spot_GP.print_results()
```

```
min y: 0.7321970236268491
x0: 9.599259898353855
x1: 3.0602975470229734

[['x0', 9.599259898353855], ['x1', 3.0602975470229734]]
```

12.7 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 0x16abc59c0>
```

```
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]]
```

12.7.1 Optimize on Surrogate

12.7.2 Evaluate on Real Objective

12.7.3 Impute / Infill new Points

12.8 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]]
```

12.9 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)}
```

12.10 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")
```

12.11 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])}
```

12.12 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")
```

12.13 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
```

```
-11.881460973212143
```

```
# vars(S)
```

```
# vars(Sf)
```

13 Hyperparameter Tuning and Noise

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

13.1 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_b4a7f5bf-ab7d-4511-9551-9bdfcf53422e_2023-06-18_22-56-36
```

13.1.1 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 0x17deeb7f0>
```

13.2 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 13.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

13.3 Noise and Surrogates: The Nugget Effect

13.3.1 The Noisy Sphere

13.3.1.1 The Data

- We prepare some data first:

```python
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:

```python
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.

13.4 Exercises

13.4.1 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])
```

13.4.2 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}
```

13.4.3 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}
```

13.4.4 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}
```

14 Handling Noise: Optimal Computational Budget Allocation in Spot

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

14.1 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
```

14.1.1 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 0x157eef670>
```

14.2 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)
```

14.3 Noise and Surrogates: The Nugget Effect

14.3.1 The Noisy Sphere

14.3.1.1 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.

14.4 Exercises

14.4.1 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])
```

14.4.2 `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}
```

14.4.3 `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}
```

14.4.4 `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
```

15 HPT: sklearn SVC on Moons Data

This document refers to the following software versions:

- `python`: 3.10.10

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

```
spotPython                  0.2.38
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
```

```
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_b4a7f5bf-ab7d-4511-9551-9bdfcf53422e_1min_5init_2023-06-18_22-58-13
```

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/10_spot_hpt_sklearn_classification")
```

15.3 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})
```

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:

```
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(),
#     )
```

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:

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

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

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

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

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

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

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

15.6 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 19.6.

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:

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

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

15.6.2 Modify hyperparameter of type factor

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

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

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

15.6.3 Optimizers

Optimizers are described in Section 19.6.1.

15.7 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,
               })
```

15.7.1 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,
               })
```

15.8 Step 8: Calling the SPOT Function

15.8.1 Preparing the SPOT Call

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

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

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

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

15.8.2 The Objective Function

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

```
from spotPython.fun.hypersklearn import HyperSklearn
fun = HyperSklearn().fun_sklearn
```

15.8.3 Run the `Spot` Optimizer

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

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

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

15.8.4 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 [----------] 2.79%

spotPython tuning: 5.691103166702708 [----------] 4.68%

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

spotPython tuning: 5.691103166702708 [#---------] 7.76%
```

```
spotPython tuning: 5.691103166702708 [#---------] 9.19%

spotPython tuning: 5.691103166702708 [#---------] 11.60%

spotPython tuning: 5.691103166702708 [#---------] 14.01%

spotPython tuning: 5.691103166702708 [##--------] 16.21%

spotPython tuning: 5.691103166702708 [##--------] 18.50%

spotPython tuning: 5.691103166702708 [##--------] 20.87%

spotPython tuning: 5.691103166702708 [##--------] 23.46%

spotPython tuning: 5.691103166702708 [###-------] 26.17%

spotPython tuning: 5.691103166702708 [###-------] 34.63%

spotPython tuning: 5.691103166702708 [#####-----] 46.64%

spotPython tuning: 5.691103166702708 [######----] 59.60%

spotPython tuning: 5.691103166702708 [#######---] 72.79%

spotPython tuning: 5.691103166702708 [#########-] 85.99%

spotPython tuning: 5.691103166702708 [##########] 98.89%

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

<spotPython.spot.spot.Spot at 0x17dd72c50>
```

15.9 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 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 |   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
```

15.9.1 Show variable importance

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

Figure 15.2: Variable importance plot, threshold 0.025.

15.9.2 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))])
```

15.9.3 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]]
```

```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': 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))])
```

15.9.4 Plot: Compare Predictions

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

```
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)
```

```
(5.691103166702708, 9.485171944504513)
```

15.9.5 Detailed Hyperparameter Plots

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

```
C:  1.1399176173997725
kernel:  100.0
```

15.9.6 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
```

15.9.7 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)
```

16 HPT: PyTorch With fashionMNIST

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

This document refers to the following software versions:

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

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

```
spotPython                  0.2.38
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, initial design size and the device that is used.

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

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

Note: Device selection

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

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

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

```
cpu
```

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

```
11-torch_b4a7f5bf-ab7d-4511-9551-9bdfcf53422e_1min_5init_2023-06-18_23-05-28
```

16.2 Step 2: Initialization of the Empty `fun_control` Dictionary

`spotPython` uses a Python dictionary for storing the information required for the hyperparameter tuning process, which was described in Section 19.2.

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

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

16.3 Step 3: PyTorch Data Loading

16.3.1 Load fashionMNIST Data

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

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

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

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

16.4 Step 4: Specification of the Preprocessing Model

After the training and test data are specified and added to the `fun_control` dictionary, `spotPython` allows the specification of a data preprocessing pipeline, e.g., for the scaling of the data or for the one-hot encoding of categorical variables, see Section 19.4. This feature is not used here, so we do not change the default value (which is `None`).

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

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

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


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

```
            k_folds=k_folds,
            patience=patience,
            optimizer=optimizer,
            sgd_momentum=sgd_momentum,
        )
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28 * 28, l1),
            nn.ReLU(),
            nn.Linear(l1, l2),
            nn.ReLU(),
            nn.Linear(l2, 10)
        )

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

This class inherits from the class `Net_Core` which is implemented in the file `netcore.py`, see Section 19.5.1.

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

16.5.1 The Search Space

16.5.2 Configuring the Search Space With spotPython

16.5.2.1 The `hyper_dict` Hyperparameters for the Selected Algorithm

`spotPython` uses `JSON` files for the specification of the hyperparameters, which were described in Section 19.5.5.

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

```python
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',
  'core_model_parameter_type': 'str',
  'lower': 0,
  'upper': 12},
 'sgd_momentum': {'type': 'float',
  'default': 0.0,
  'transform': 'None',
  'lower': 0.0,
  'upper': 1.0}}
```

16.6 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 19.6.

16.6.1 Modify hyperparameter of type numeric and integer (boolean)

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

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

- Net sizes `l1` and `l2` as well as `epochs` and `patience` are set to small values for demonstration purposes. These values are too small for a real application.
- More resonable values are, e.g.:
  - `fun_control = modify_hyper_parameter_bounds(fun_control, "l1", bounds=[2, 7])`
  - `fun_control = modify_hyper_parameter_bounds(fun_control,`

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

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

16.6.2 Modify hyperparameter of type factor

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

16.6.3 Optimizers

Optimizers are described in Section 19.6.1.

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

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

16.7.1 Evaluation

The evaluation procedure requires the specification of two elements:

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

These are described in Section 24.7.1.

The key `"loss_function"` specifies the loss function which is used during the optimization, see Section 19.7.5.

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

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

16.7.2 Metric

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

16.8 Step 8: Calling the SPOT Function

16.8.1 Preparing the SPOT Call

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

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

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

```
| name          | type    | default    |   lower |    upper | transform              |
|---------------|---------|------------|---------|----------|------------------------|
```

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

16.8.2 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
```

16.8.3 Starting the Hyperparameter Tuning

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

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

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

MulticlassAccuracy: 0.1052500009536743 | Loss: 2.2871348048845928 | Acc: 0.1052500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.1814583390951157 | Loss: 2.2488501973152162 | Acc: 0.1814583333333333.
Epoch: 3 |

MulticlassAccuracy: 0.2147916704416275 | Loss: 2.2160350697835285 | Acc: 0.2147916666666667.
Epoch: 4 |

MulticlassAccuracy: 0.2313749939203262 | Loss: 2.1862338910102843 | Acc: 0.2313750000000000.
Epoch: 5 |

MulticlassAccuracy: 0.2420416623353958 | Loss: 2.1586866825421653 | Acc: 0.2420416666666667.
Epoch: 6 |

MulticlassAccuracy: 0.2485000044107437 | Loss: 2.1326535865465801 | Acc: 0.2485000000000000.
Epoch: 7 |

MulticlassAccuracy: 0.2532083392143250 | Loss: 2.1079019922415414 | Acc: 0.2532083333333333.
Epoch: 8 |
```

```
MulticlassAccuracy: 0.2550416588783264 | Loss: 2.0843258651892342 | Acc: 0.2550416666666667.
Returned to Spot: Validation loss: 2.0843258651892342

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


MulticlassAccuracy: 0.1457083374261856 | Loss: 2.2969474332729973 | Acc: 0.1457083333333333.
Epoch: 2 |


MulticlassAccuracy: 0.2241249978542328 | Loss: 2.2766560192108156 | Acc: 0.2241250000000000.
Epoch: 3 |


MulticlassAccuracy: 0.2617916762828827 | Loss: 2.2589634562333423 | Acc: 0.2617916666666666.
Epoch: 4 |


MulticlassAccuracy: 0.2800416648387909 | Loss: 2.2420300889412563 | Acc: 0.2800416666666667.
Returned to Spot: Validation loss: 2.2420300889412563

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


MulticlassAccuracy: 0.2886666655540466 | Loss: 2.1501315731008845 | Acc: 0.2886666666666667.
Epoch: 2 |


MulticlassAccuracy: 0.2623333334922791 | Loss: 1.9485283993184566 | Acc: 0.2623333333333333.
Epoch: 3 |


MulticlassAccuracy: 0.3241666555404663 | Loss: 1.7340658128112554 | Acc: 0.3241666666666667.
Epoch: 4 |


MulticlassAccuracy: 0.4793333411216736 | Loss: 1.5400967248330513 | Acc: 0.4793333333333333.
Epoch: 5 |


MulticlassAccuracy: 0.5345000028610229 | Loss: 1.3770331937174003 | Acc: 0.5345000000000000.
Epoch: 6 |


MulticlassAccuracy: 0.5770000219345093 | Loss: 1.2462426517295340 | Acc: 0.5770000000000000.
Epoch: 7 |
```

```
MulticlassAccuracy: 0.6412500143051147 | Loss: 1.1403913558845720 | Acc: 0.6412500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.6736249923706055 | Loss: 1.0512866181274876 | Acc: 0.6736250000000000.
Returned to Spot: Validation loss: 1.0512866181274876

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

MulticlassAccuracy: 0.1170416697859764 | Loss: 2.3122467995683351 | Acc: 0.1170416666666667.
Epoch: 2 |

MulticlassAccuracy: 0.1637083292007446 | Loss: 2.3061220555702846 | Acc: 0.1637083333333333.
Epoch: 3 |

MulticlassAccuracy: 0.1803749948740005 | Loss: 2.2997353075146676 | Acc: 0.1803750000000000.
Epoch: 4 |

MulticlassAccuracy: 0.1839583367109299 | Loss: 2.2929599477450053 | Acc: 0.1839583333333333.
Returned to Spot: Validation loss: 2.2929599477450053

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

MulticlassAccuracy: 0.1032499969005585 | Loss: 2.2967566425005597 | Acc: 0.1032500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.1038749963045120 | Loss: 2.2678986296653747 | Acc: 0.1038750000000000.
Epoch: 3 |

MulticlassAccuracy: 0.1034166663885117 | Loss: 2.2377995806535087 | Acc: 0.1034166666666667.
Epoch: 4 |

MulticlassAccuracy: 0.1209583356976509 | Loss: 2.2042476224104561 | Acc: 0.1209583333333333.
Epoch: 5 |

MulticlassAccuracy: 0.2127083390951157 | Loss: 2.1656834946076073 | Acc: 0.2127083333333333.
Epoch: 6 |
```

```
MulticlassAccuracy: 0.2801666557788849 | Loss: 2.1235535779794059 | Acc: 0.2801666666666667.
Epoch: 7 |

MulticlassAccuracy: 0.2977083325386047 | Loss: 2.0781978339354197 | Acc: 0.2977083333333334.
Epoch: 8 |

MulticlassAccuracy: 0.3069166541099548 | Loss: 2.0298420123656591 | Acc: 0.3069166666666667.
Returned to Spot: Validation loss: 2.029842012365659


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

MulticlassAccuracy: 0.2439166605472565 | Loss: 2.2502474977175395 | Acc: 0.2439166666666667.
Epoch: 2 |

MulticlassAccuracy: 0.2430833280086517 | Loss: 2.1840287584066389 | Acc: 0.2430833333333333.
Epoch: 3 |

MulticlassAccuracy: 0.2749166786670685 | Loss: 2.1204189803600313 | Acc: 0.2749166666666666.
Epoch: 4 |

MulticlassAccuracy: 0.3219583332538605 | Loss: 2.0592026861508685 | Acc: 0.3219583333333333.
Epoch: 5 |

MulticlassAccuracy: 0.3564166724681854 | Loss: 1.9988209830125172 | Acc: 0.3564166666666667.
Epoch: 6 |

MulticlassAccuracy: 0.3738333284854889 | Loss: 1.9393671920696893 | Acc: 0.3738333333333334.
Epoch: 7 |

MulticlassAccuracy: 0.3863750100135803 | Loss: 1.8811635773976645 | Acc: 0.3863750000000000.
Epoch: 8 |

MulticlassAccuracy: 0.4166666567325592 | Loss: 1.8238327132860819 | Acc: 0.4166666666666667.
Returned to Spot: Validation loss: 1.8238327132860819
spotPython tuning: 1.0512866181274876 [#####-----] 45.72%
```

```
config: {'l1': 16, 'l2': 16, 'lr_mult': 0.001, 'batch_size': 2, 'epochs': 4, 'k_folds': 0, 'p
Epoch: 1 |

MulticlassAccuracy: 0.1615000069141388 | Loss: 2.2676606412827969 | Acc: 0.1615000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.2156666666269302 | Loss: 2.2166375264922777 | Acc: 0.2156666666666667.
Epoch: 3 |

MulticlassAccuracy: 0.2269999980926514 | Loss: 2.1549972802400590 | Acc: 0.2270000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.2329583317041397 | Loss: 2.0891105142831803 | Acc: 0.2329583333333333.
Returned to Spot: Validation loss: 2.0891105142831803

spotPython tuning: 1.0512866181274876 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x17fdbfb20>
```

16.9 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section 19.9, see also the description in the documentation: Tensorboard.

16.10 Step 10: Results

After the hyperparameter tuning run is finished, the results can be analyzed as described in Section 19.10.

```
SAVE = False
LOAD = False

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

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

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

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

Figure 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              |
|---------------|---------|------------|----------|----------|----------|------------------------|--
| l1            | int     | 5          |      2.0 |      5.0 |      5.0 | transform_power_2_int  |
| l2            | int     | 5          |      2.0 |      5.0 |      4.0 | transform_power_2_int  |
| lr_mult       | float   | 1.0        |    0.001 |    0.001 |    0.001 | None                   |
| batch_size    | int     | 4          |      1.0 |      4.0 |      1.0 | transform_power_2_int  |
| epochs        | int     | 3          |      2.0 |      3.0 |      3.0 | transform_power_2_int  |
| k_folds       | int     | 1          |      0.0 |      0.0 |      0.0 | None                   |
| patience      | int     | 5          |      2.0 |      2.0 |      2.0 | None                   |
| optimizer     | factor  | SGD        |      0.0 |      3.0 |      3.0 | None                   |
| sgd_momentum  | float   | 0.0        |      0.9 |      0.9 |      0.9 | None                   |
```

16.10.1 Show variable importance

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

Figure 16.2: Variable importance plot, threshold 0.025.

16.10.2 Get the Tuned Architecture (SPOT Results)

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

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

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

```
  )
)
```

16.10.3 Get Default Hyperparameters

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

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

16.10.4 Evaluation of the Default and the Tuned Architectures

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

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

```
Epoch: 1 |
```

```
MulticlassAccuracy: 0.3656666576862335 | Loss: 1.9611297413508098 | Acc: 0.3656666666666666.
Epoch: 2 |

MulticlassAccuracy: 0.5431666374206543 | Loss: 1.5176253639062245 | Acc: 0.5431666666666667.
Epoch: 3 |

MulticlassAccuracy: 0.5835000276565552 | Loss: 1.2576823236147563 | Acc: 0.5835000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.6041250228881836 | Loss: 1.1103421303431193 | Acc: 0.6041250000000000.
Epoch: 5 |

MulticlassAccuracy: 0.6282500028610229 | Loss: 1.0193856404225032 | Acc: 0.6282500000000000.
Epoch: 6 |

MulticlassAccuracy: 0.6504166722297668 | Loss: 0.9576737509767215 | Acc: 0.6504166666666666.
Epoch: 7 |

MulticlassAccuracy: 0.6683333516120911 | Loss: 0.9135884566505750 | Acc: 0.6683333333333333.
Epoch: 8 |

MulticlassAccuracy: 0.6837499737739563 | Loss: 0.8781539583007495 | Acc: 0.6837500000000000.
Returned to Spot: Validation loss: 0.8781539583007495
```

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

```
MulticlassAccuracy: 0.6683999896049500 | Loss: 0.8914014785289764 | Acc: 0.6684000000000000.
Final evaluation: Validation loss: 0.8914014785289764
Final evaluation: Validation metric: 0.66839998960495
---------------------------------------------

(0.8914014785289764, nan, tensor(0.6684))
```

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

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

```
Epoch: 1 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.252

MulticlassAccuracy: 0.3937083184719086 | Loss: 2.0882351970275241 | Acc: 0.3937083333333333.
Epoch: 2 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.022

MulticlassAccuracy: 0.3524166643619537 | Loss: 1.8519588796943427 | Acc: 0.3524166666666667.
Epoch: 3 |

Batch: 10000. Batch Size: 2. Training Loss (running): 1.782

MulticlassAccuracy: 0.4505416750907898 | Loss: 1.6164510637608667 | Acc: 0.4505416666666667.
Epoch: 4 |

Batch: 10000. Batch Size: 2. Training Loss (running): 1.548

MulticlassAccuracy: 0.5301250219345093 | Loss: 1.4011302709206939 | Acc: 0.5301250000000000.
Epoch: 5 |

Batch: 10000. Batch Size: 2. Training Loss (running): 1.343

MulticlassAccuracy: 0.5645416378974915 | Loss: 1.2275900119623790 | Acc: 0.5645416666666667.
Epoch: 6 |

Batch: 10000. Batch Size: 2. Training Loss (running): 1.177
```

```
MulticlassAccuracy: 0.6051250100135803 | Loss: 1.0942692651928714 | Acc: 0.6051250000000000.
Epoch: 7 |

Batch: 10000. Batch Size: 2. Training Loss (running): 1.054

MulticlassAccuracy: 0.6404583454132080 | Loss: 1.0004381538924452 | Acc: 0.6404583333333334.
Epoch: 8 |

Batch: 10000. Batch Size: 2. Training Loss (running): 0.973

MulticlassAccuracy: 0.6618750095367432 | Loss: 0.9358425689215462 | Acc: 0.6618750000000000.
Returned to Spot: Validation loss: 0.9358425689215462
```

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

```
MulticlassAccuracy: 0.6541000008583069 | Loss: 0.9456034678556025 | Acc: 0.6541000000000000.
Final evaluation: Validation loss: 0.9456034678556025
Final evaluation: Validation metric: 0.6541000008583069
-----------------------------------------------

(0.9456034678556025, nan, tensor(0.6541))
```

16.10.5 Detailed Hyperparameter Plots

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

```
l1:  2.4756053654746224
optimizer:  100.0
```

Figure 16.3: Contour plots.

16.10.6 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

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

Parallel coordinates plots

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

16.10.7 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

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

17 HPT: PyTorch With cifar10 Data

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

This document refers to the following software versions:

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

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

```
spotPython                  0.2.38
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
```

17.1 Step 1: Setup

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

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

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

Note: Device selection

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

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

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

```
cpu
```

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

```
12-torch_bartz09_1min_5init_2023-06-18_23-22-46
```

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

`spotPython` uses a Python dictionary for storing the information required for the hyperparameter tuning process, which was described in Section 19.2.

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

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

17.3 Step 3: PyTorch Data Loading

17.3.1 Load Data Cifar10 Data

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

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

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

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

```
Files already downloaded and verified
```

```
Files already downloaded and verified
```

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

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

17.4 Step 4: Specification of the Preprocessing Model

After the training and test data are specified and added to the `fun_control` dictionary, `spotPython` allows the specification of a data preprocessing pipeline, e.g., for the scaling of the data or for the one-hot encoding of categorical variables, see Section 19.4. This feature is not used here, so we do not change the default value (which is `None`).

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

17.5.1 Implementing a Configurable Neural Network With spotPython

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

This class inherits from the class `Net_Core` which is implemented in the file `netcore.py`, see Section 19.5.1.

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

17.5.2 The Search Space

17.5.3 Configuring the Search Space With spotPython

17.5.3.1 The `hyper_dict` Hyperparameters for the Selected Algorithm

`spotPython` uses `JSON` files for the specification of the hyperparameters, which were described in Section 19.5.5.

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

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

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

```
  'upper': 1},
 'patience': {'type': 'int',
  'default': 5,
  'transform': 'None',
  'lower': 2,
  'upper': 10},
 'optimizer': {'levels': ['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}}
```


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 19.6.

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

17.6.1.1 Modify Hyperparameters of Type numeric and integer (boolean)

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

> Caution: Small net size, number of epochs, and patience for demonstration purposes
>
> - Net sizes `l1` and `l2` as well as `epochs` and `patience` are set to small values for demonstration purposes. These values are too small for a real application.
> - More resonable values are, e.g.:
> - `fun_control = modify_hyper_parameter_bounds(fun_control, "l1", bounds=[2, 7])`
> - `fun_control = modify_hyper_parameter_bounds(fun_control, "epochs", bounds=[7, 9])` and
> - `fun_control = modify_hyper_parameter_bounds(fun_control, "patience", bounds=[2, 7])`

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

17.6.2 Modify hyperparameter of type factor

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

17.6.3 Optimizers

Optimizers can be selected as described in Section 24.6.2.

Optimizers are described in Section 19.6.1.

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

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

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 and
2. the loss function (and a metric).

These are described in Section 24.7.1.

The key `"loss_function"` specifies the loss function which is used during the optimization, see Section 19.7.5.

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

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

17.7.2 Metric

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

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`.

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

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

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

17.8.2 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
```

17.8.3 Starting the Hyperparameter Tuning

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

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

MulticlassAccuracy: 0.0999500006437302 | Loss: 2.3179945016860963 | Acc: 0.0999500000000000.
Epoch: 2 |
```

```
MulticlassAccuracy: 0.0999500006437302 | Loss: 2.3176324953079224 | Acc: 0.0999500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.0999500006437302 | Loss: 2.3172526622772218 | Acc: 0.0999500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.0999500006437302 | Loss: 2.3168686109542849 | Acc: 0.0999500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.0999500006437302 | Loss: 2.3164956390380858 | Acc: 0.0999500000000000.
Epoch: 6 |

MulticlassAccuracy: 0.0999500006437302 | Loss: 2.3161151149749757 | Acc: 0.0999500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.0999500006437302 | Loss: 2.3156693559646606 | Acc: 0.0999500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.0999500006437302 | Loss: 2.3150685266494753 | Acc: 0.0999500000000000.
Returned to Spot: Validation loss: 2.3150685266494753

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

MulticlassAccuracy: 0.1028999984264374 | Loss: 2.3272440153121949 | Acc: 0.1029000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.1032499969005585 | Loss: 2.3263748434066773 | Acc: 0.1032500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.1027500033378601 | Loss: 2.3258572361946106 | Acc: 0.1027500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.1032000035047531 | Loss: 2.3254805560112000 | Acc: 0.1032000000000000.
Returned to Spot: Validation loss: 2.3254805560112

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

```
MulticlassAccuracy: 0.0988000035285950 | Loss: 2.3020611174821854 | Acc: 0.0988000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.1459500044584274 | Loss: 2.2745127540946006 | Acc: 0.1459500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.1650499999523163 | Loss: 2.2383212146997451 | Acc: 0.1650500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.1768500059843063 | Loss: 2.2078618783831598 | Acc: 0.1768500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.1906500011682510 | Loss: 2.1837500626862050 | Acc: 0.1906500000000000.
Epoch: 6 |

MulticlassAccuracy: 0.2039500027894974 | Loss: 2.1608802776575087 | Acc: 0.2039500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.2126500010490417 | Loss: 2.1386569876730444 | Acc: 0.2126500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.2188500016927719 | Loss: 2.1152741783618927 | Acc: 0.2188500000000000.
Returned to Spot: Validation loss: 2.1152741783618927

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

MulticlassAccuracy: 0.0971499979496002 | Loss: 2.3285051279306410 | Acc: 0.0971500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.0971499979496002 | Loss: 2.3274505632400513 | Acc: 0.0971500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.0971499979496002 | Loss: 2.3262843572616578 | Acc: 0.0971500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.0971499979496002 | Loss: 2.3249345053672790 | Acc: 0.0971500000000000.
Returned to Spot: Validation loss: 2.324934505367279

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

```
MulticlassAccuracy: 0.0974000021815300 | Loss: 2.3102536315917970 | Acc: 0.0974000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.0974000021815300 | Loss: 2.3079652256011962 | Acc: 0.0974000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.0974000021815300 | Loss: 2.3048800094604491 | Acc: 0.0974000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.0974000021815300 | Loss: 2.3010525613784791 | Acc: 0.0974000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.0974000021815300 | Loss: 2.2970885346412659 | Acc: 0.0974000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.0974000021815300 | Loss: 2.2928804782867434 | Acc: 0.0974000000000000.
Epoch: 7 |

MulticlassAccuracy: 0.0974999964237213 | Loss: 2.2881885060310365 | Acc: 0.0975000000000000.
Epoch: 8 |

MulticlassAccuracy: 0.0987500026822090 | Loss: 2.2828654908180237 | Acc: 0.0987500000000000.
Returned to Spot: Validation loss: 2.2828654908180237

config: {'l1': 8, 'l2': 16, 'lr_mult': 0.001, 'batch_size': 8, 'epochs': 8, 'k_folds': 0, 'pa

Epoch: 1 |

MulticlassAccuracy: 0.1011999994516373 | Loss: 2.3197656811714173 | Acc: 0.1012000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.1011999994516373 | Loss: 2.3156074615478515 | Acc: 0.1012000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.1008500009775162 | Loss: 2.3117004761695861 | Acc: 0.1008500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.1005000025033951 | Loss: 2.3076723896980287 | Acc: 0.1005000000000000.
Epoch: 5 |
```

```
MulticlassAccuracy: 0.1015999987721443 | Loss: 2.3032591325759886 | Acc: 0.1016000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.1062500029802322 | Loss: 2.2982188761711120 | Acc: 0.1062500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.1149500012397766 | Loss: 2.2922247573852541 | Acc: 0.1149500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.1302500069141388 | Loss: 2.2851447748184204 | Acc: 0.1302500000000000.
Returned to Spot: Validation loss: 2.2851447748184204
spotPython tuning: 2.1152741783618927 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x2b61cfb50>
```

17.9 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section 19.9, see also the description in the documentation: Tensorboard.

17.10 Step 10: Results

After the hyperparameter tuning run is finished, the results can be analyzed as described in Section 19.10.

```
SAVE = False
LOAD = False

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

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

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

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

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

- Print the results

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

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

17.10.1 Show variable importance

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

Figure 17.2: Variable importance plot, threshold 0.025.

17.10.2 Get the Tuned Architecture (SPOT Results)

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

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

```
Net_CIFAR10(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=32, bias=True)
  (fc2): Linear(in_features=32, out_features=16, bias=True)
  (fc3): Linear(in_features=16, out_features=10, bias=True)
)
```

17.10.3 Evaluation of the Tuned Architecture

```
from spotPython.torch.traintest import (
    train_tuned,
    test_tuned,
    )
```

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

```
Epoch: 1 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.304

MulticlassAccuracy: 0.1195499971508980 | Loss: 2.2925230521678923 | Acc: 0.1195500000000000.
Epoch: 2 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.283

MulticlassAccuracy: 0.1307999938726425 | Loss: 2.2584577577829359 | Acc: 0.1308000000000000.
Epoch: 3 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.243

MulticlassAccuracy: 0.1393000036478043 | Loss: 2.2134238643527029 | Acc: 0.1393000000000000.
Epoch: 4 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.200

MulticlassAccuracy: 0.1628500074148178 | Loss: 2.1702615287423135 | Acc: 0.1628500000000000.
Epoch: 5 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.158
```

```
MulticlassAccuracy: 0.2089499980211258 | Loss: 2.1327888869404794 | Acc: 0.2089500000000000.
Epoch: 6 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.121

MulticlassAccuracy: 0.2379499971866608 | Loss: 2.1031303014278411 | Acc: 0.2379500000000000.
Epoch: 7 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.092

MulticlassAccuracy: 0.2473500072956085 | Loss: 2.0793005030274392 | Acc: 0.2473500000000000.
Epoch: 8 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.077

MulticlassAccuracy: 0.2526000142097473 | Loss: 2.0601810956478119 | Acc: 0.2526000000000000.
Returned to Spot: Validation loss: 2.060181095647812
```

If `path` is set to a filename, e.g., `path = "model_spot_trained.pt"`, the weights of the trained model will be loaded from this file.

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

```
MulticlassAccuracy: 0.2594999969005585 | Loss: 2.0521580039143563 | Acc: 0.2595000000000000.
Final evaluation: Validation loss: 2.0521580039143563
Final evaluation: Validation metric: 0.25949999690055847
---------------------------------------------

(2.0521580039143563, nan, tensor(0.2595))
```

17.10.4 Cross-validated Evaluations

Caution: Cross-validated Evaluations

- The number of folds is set to 1 by default.
- Here it was changed to 3 for demonstration purposes.
- Set the number of folds to a reasonable value, e.g., 10.
- This can be done by setting the `k_folds` attribute of the model as follows:
- `setattr(model_spot, "k_folds",  10)`

```
from spotPython.torch.traintest import evaluate_cv
# modify k-kolds:
setattr(model_spot, "k_folds",  3)
df_eval, df_preds, df_metrics = evaluate_cv(net=model_spot,
            dataset=fun_control["data"],
            loss_function=fun_control["loss_function"],
            metric=fun_control["metric_torch"],
            task=fun_control["task"],
            writer=fun_control["writer"],
            writerId="model_spot_cv",
            device = fun_control["device"])
```

```
Error in Net_Core. Call to evaluate_cv() failed. err=TypeError("Expected sequence or array-li
```

```
metric_name = type(fun_control["metric_torch"]).__name__
print(f"loss: {df_eval}, Cross-validated {metric_name}: {df_metrics}")
```

```
loss: nan, Cross-validated MulticlassAccuracy: nan
```

17.10.5 Detailed Hyperparameter Plots

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

```
l1:  96.48026604244428
batch_size:  100.0
optimizer:  0.9076449353269873
```

Figure 17.3: Contour plots.

17.10.6 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

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

Parallel coordinates plots

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

17.10.7 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

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

18 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.

18.1 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_bartz09_1min_5init_2023-06-18_23-37-13
```

18.1.1 `river` Hyperparameter Tuning: HATR with Friedman Drift Data

- This notebook exemplifies hyperparameter tuning with SPOT (spotPython and spotRiver).
- The hyperparameter software SPOT was developed in R (statistical programming language), see Open Access book "Hyperparameter Tuning for Machine and Deep Learning with R - A Practical Guide", available here: https://link.springer.com/book/10.1007/978-981-19-5170-1.

- This notebook demonstrates hyperparameter tuning for `river`. It is based on the notebook "Incremental decision trees in river: the Hoeffding Tree case", see: https://riverml.xyz/0.15.0/recipes/on-hoeffding-trees/#42-regression-tree-splitters.
- Here we will use the river `HTR` and `HATR` functions as in "Incremental decision trees in river: the Hoeffding Tree case", see: https://riverml.xyz/0.15.0/recipes/on-hoeffding-trees/#42-regression-tree-splitters.

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

```
spotPython                  0.2.38
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
```

18.2 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)
```

18.3 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})
```

18.4 Step 4: Specification of the Preprocessing Model

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

18.5 Step 5: Select `algorithm` and `core_model_hyper_dict`

- The `river` model (`HATR`) is selected.
- Furthermore, the corresponding hyperparameters, see: https://riverml.xyz/0.15.0/api/tree/HoeffdingTreeRegressor/ are selected (incl. type information, names, and bounds).
- The corresponding hyperparameter dictionary is added to the `fun_control` dictionary.
- Alternatively, you can load a local hyper_dict. Simply set `river_hyper_dict.json` as the filename. If `filename`is set to `None`, the hyper_dict is loaded from the `spotRiver` package.

```
from river.tree import HoeffdingAdaptiveTreeRegressor
from spotRiver.data.river_hyper_dict import RiverHyperDict
from spotPython.hyperparameters.values import add_core_model_to_fun_control
core_model  = HoeffdingAdaptiveTreeRegressor
fun_control = add_core_model_to_fun_control(core_model=core_model,
                              fun_control=fun_control,
                              hyper_dict=RiverHyperDict,
                              filename=None)
```

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

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

```
{'grace_period': {'type': 'int',
  'default': 200,
  'transform': 'None',
  'lower': 10,
  'upper': 1000},
 'max_depth': {'type': 'int',
  'default': 20,
  'transform': 'transform_power_2_int',
  'lower': 2,
  'upper': 20},
 'delta': {'type': 'float',
  'default': 1e-07,
  'transform': 'None',
  'lower': 1e-08,
  'upper': 1e-06},
 'tau': {'type': 'float',
  'default': 0.05,
  'transform': 'None',
  'lower': 0.01,
```

```
 'upper': 0.1},
'leaf_prediction': {'levels': ['mean', 'model', 'adaptive'],
 'type': 'factor',
 'default': 'mean',
 'transform': 'None',
 'core_model_parameter_type': 'str',
 'lower': 0,
 'upper': 2},
'leaf_model': {'levels': ['LinearRegression', 'PARegressor', 'Perceptron'],
 'type': 'factor',
 'default': 'LinearRegression',
 'transform': 'None',
 'class_name': 'river.linear_model',
 'core_model_parameter_type': 'instance()',
 'lower': 0,
 'upper': 2},
'model_selector_decay': {'type': 'float',
 'default': 0.95,
 'transform': 'None',
 'lower': 0.9,
 'upper': 0.99},
'splitter': {'levels': ['EBSTSplitter', 'TEBSTSplitter', 'QOSplitter'],
 'type': 'factor',
 'default': 'EBSTSplitter',
 'transform': 'None',
 'class_name': 'river.tree.splitter',
 'core_model_parameter_type': 'instance()',
 'lower': 0,
 'upper': 2},
'min_samples_split': {'type': 'int',
 'default': 5,
 'transform': 'None',
 'lower': 2,
 'upper': 10},
'bootstrap_sampling': {'levels': [0, 1],
 'type': 'factor',
 'default': 0,
 'transform': 'None',
 'core_model_parameter_type': 'bool',
 'lower': 0,
 'upper': 1},
'drift_window_threshold': {'type': 'int',
 'default': 300,
```

```
 'transform': 'None',
 'lower': 100,
 'upper': 500},
'switch_significance': {'type': 'float',
 'default': 0.05,
 'transform': 'None',
 'lower': 0.01,
 'upper': 0.1},
'binary_split': {'levels': [0, 1],
 'type': 'factor',
 'default': 0,
 'transform': 'None',
 'core_model_parameter_type': 'bool',
 'lower': 0,
 'upper': 1},
'max_size': {'type': 'float',
 'default': 500.0,
 'transform': 'None',
 'lower': 100.0,
 'upper': 1000.0},
'memory_estimate_period': {'type': 'int',
 'default': 1000000,
 'transform': 'None',
 'lower': 100000,
 'upper': 1000000},
'stop_mem_management': {'levels': [0, 1],
 'type': 'factor',
 'default': 0,
 'transform': 'None',
 'core_model_parameter_type': 'bool',
 'lower': 0,
 'upper': 1},
'remove_poor_attrs': {'levels': [0, 1],
 'type': 'factor',
 'default': 0,
 'transform': 'None',
 'core_model_parameter_type': 'bool',
 'lower': 0,
 'upper': 1},
'merit_preprune': {'levels': [0, 1],
 'type': 'factor',
 'default': 0,
 'transform': 'None',
```

```
'core_model_parameter_type': 'bool',
'lower': 0,
'upper': 1}}
```

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

18.6.1 Modify hyperparameter of type factor

```
# fun_control = modify_hyper_parameter_levels(fun_control, "leaf_model", ["LinearRegressio
# fun_control["core_model_hyper_dict"]
```

18.6.2 Modify hyperparameter of type numeric and integer (boolean)

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

18.7 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
                })
```

18.8 Step 8: Calling the SPOT Function

18.8.1 Prepare the SPOT Parameters

- Get types and variable names as well as lower and upper bounds for the hyperparameters.

```
from spotPython.hyperparameters.values import (
    get_var_type,
    get_var_name,
    get_bound_values
    )
var_type = get_var_type(fun_control)
var_name = get_var_name(fun_control)
fun_control.update({"var_type": var_type,
                    "var_name": var_name})

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

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

```
| name                    | type    | default           |       lower |     upper | transform
|-------------------------|---------|-------------------|-------------|-----------|---------------
```

```
| grace_period           | int    | 200              |     10     | 1000     | None
| max_depth              | int    | 20               |      2     |   20     | transform_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
```

18.8.2 Run the `Spot` Optimizer

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

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

```
from spotPython.spot import spot
from math import inf
import numpy as np
spot_tuner = spot.Spot(fun=fun,
                    lower = lower,
                    upper = upper,
                    fun_evals = inf,
                    fun_repeats = 1,
                    max_time = MAX_TIME,
                    noise = False,
                    tolerance_x = np.sqrt(np.spacing(1)),
                    var_type = var_type,
```

```
                    var_name = var_name,
                    infill_criterion = "y",
                    n_points = 1,
                    seed=123,
                    log_level = 50,
                    show_models= False,
                    show_progress= True,
                    fun_control = fun_control,
                    design_control={"init_size": INIT_SIZE,
                                    "repeats": 1},
                    surrogate_control={"noise": True,
                                       "cod_type": "norm",
                                       "min_theta": -4,
                                       "max_theta": 3,
                                       "n_theta": len(var_name),
                                       "model_fun_evals": 10_000,
                                       "log_level": 50
                                       })
spot_tuner.run(X_start=X_start)
```

```
spotPython tuning: 2.17160762076232 [#---------] 13.46%

spotPython tuning: 2.17160762076232 [###-------] 26.62%

spotPython tuning: 2.17160762076232 [#####-----] 49.98%

spotPython tuning: 2.1460829608878695 [#######---] 68.37%

spotPython tuning: 2.1460829608878695 [#########-] 93.15%

spotPython tuning: 2.1437499252562224 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x28bf92f50>
```

18.9 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 |              643
| max_depth              | int    | 20               |      2.0 |      20.0 |               19
| delta                  | float  | 1e-07            |    1e-10 |     1e-06 |              1e-
| tau                    | float  | 0.05             |     0.01 |       0.1 |               0
| leaf_prediction        | factor | mean             |      0.0 |       2.0 |
```

```
| leaf_model              | factor | LinearRegression |       0.0 |        2.0 |                 0
| model_selector_decay    | float  | 0.95             |       0.9 |       0.99 |                0
| splitter                | factor | EBSTSplitter     |       0.0 |        2.0 |                 2
| min_samples_split       | int    | 5                |       2.0 |       10.0 |                 9
| bootstrap_sampling      | factor | 0                |       0.0 |        1.0 |                 0
| drift_window_threshold  | int    | 300              |     100.0 |      500.0 |               114
| switch_significance     | float  | 0.05             |      0.01 |        0.1 |                0
| binary_split            | factor | 0                |       0.0 |        1.0 |                 0
| max_size                | float  | 500.0            |     100.0 |     1000.0 | 667.3846405009
| memory_estimate_period  | int    | 1000000          | 100000.0 | 1000000.0 |           65855
| stop_mem_management     | factor | 0                |       0.0 |        1.0 |                 1
| remove_poor_attrs       | factor | 0                |       0.0 |        1.0 |                 0
| merit_preprune          | factor | 0                |       0.0 |        0.0 |                 0
```

18.9.1 Show variable importance

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

18.9.2 Build and Evaluate HTR Model with Tuned Hyperparameters

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

18.9.3 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"]
```

```python
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})
```

18.9.4 Get Default Hyperparameters

```python
# fun_control was modified, we generate a new one with the original
# default hyperparameters
from spotPython.hyperparameters.values import get_one_core_model_from_X
fc = fun_control
fc.update({"core_model_hyper_dict":
    hyper_dict[fun_control["core_model"].__name__]})
model_default = get_one_core_model_from_X(X_start, fun_control=fc)
model_default
```

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

```
    clip_gradient=1e+12
    initializer=Zeros ()
  )
  model_selector_decay=0.95
  nominal_attributes=None
  splitter=EBSTSplitter ()
  min_samples_split=5
  bootstrap_sampling=0
  drift_window_threshold=300
  drift_detector=ADWIN (
    delta=0.002
    clock=32
    max_buckets=5
    min_window_length=5
    grace_period=10
  )
  switch_significance=0.05
  binary_split=0
  max_size=500.
  memory_estimate_period=1000000
  stop_mem_management=0
  remove_poor_attrs=0
  merit_preprune=0
  seed=None
)
```

```
from spotRiver.evaluation.eval_bml import eval_oml_horizon

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

```
from spotRiver.evaluation.eval_bml import plot_bml_oml_horizon_metrics, plot_bml_oml_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
```

18.9.5 Get SPOT Results

```python
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=643
  max_depth=524288
  delta=1e-10
  tau=0.01
  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=0.99
  nominal_attributes=None
  splitter=QOSplitter (
    radius=0.25
    allow_multiway_splits=False
  )
  min_samples_split=9
  bootstrap_sampling=0
  drift_window_threshold=114
  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=667.384641
  memory_estimate_period=658552
  stop_mem_management=1
  remove_poor_attrs=0
  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
```

18.9.6 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}
```

18.9.7 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()
```

```python
model_spot.summary
```

```
{'n_nodes': 59,
 'n_branches': 29,
 'n_leaves': 30,
 'n_active_leaves': 29,
 'n_inactive_leaves': 0,
 'height': 13,
 'total_observed_weight': 39002.0,
 'n_alternate_trees': 21,
 'n_pruned_alternate_trees': 19,
 'n_switch_alternate_trees': 1}
```

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

```
| Parameter                 |   Default |   Spot |
|---------------------------|-----------|--------|
| n_nodes                   |        35 |     59 |
| n_branches                |        17 |     29 |
| n_leaves                  |        18 |     30 |
| n_active_leaves           |        96 |     29 |
| n_inactive_leaves         |         0 |      0 |
| height                    |         6 |     13 |
| total_observed_weight     |     39002 |  39002 |
| n_alternate_trees         |        21 |     21 |
| n_pruned_alternate_trees  |         6 |     19 |
| n_switch_alternate_trees  |         2 |      1 |
```

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

```
(2.1437499252562224, 13.363566966221045)
```

18.9.8 Detailed Hyperparameter Plots

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

```
leaf_prediction:  15.334073607770325
leaf_model:  100.0
binary_split:  0.035693776725994115
```

18.9.9 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
```

18.9.10 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)
```

19 HPT: PyTorch With spotPython and Ray Tune on CIFAR10

In this tutorial, we will show how spotPython can be integrated into the PyTorch training workflow. It is based on the tutorial "Hyperparameter Tuning with Ray Tune" from the PyTorch documentation (PyTorch 2023a), which is an extension of the tutorial "Training a Classifier" (PyTorch 2023b) for training a CIFAR10 image classifier.

This document refers to the following software versions:

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

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

```
spotPython                  0.2.38
spotRiver                   0.0.93

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

spotPython can be installed via pip[1].

```
!pip install spotPython
```

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

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

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

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

19.1 Step 1: Setup

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

> Caution: Run time and initial design size should be increased for real experiments
>
> - MAX_TIME is set to one minute for demonstration purposes. For real experiments, this should be increased to at least 1 hour.
> - INIT_SIZE is set to 5 for demonstration purposes. For real experiments, this should be increased to at least 10.

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

```
MAX_TIME = 10
INIT_SIZE = 5
DEVICE = "cpu" # "cuda:0"
```

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

```
cpu
```

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

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

```
import warnings
from datetime import datetime
from dateutil.tz import tzlocal
start_time = datetime.now(tzlocal())
HOSTNAME = socket.gethostname().split(".")[0]
experiment_name = '14-torch' + "_" + HOSTNAME + "_" + str(MAX_TIME) + "min_" + str(INIT_SI
experiment_name = experiment_name.replace(':', '-')
print(experiment_name)
if not os.path.exists('./figures'):
    os.makedirs('./figures')
warnings.filterwarnings("ignore")
```

```
14-torch_bartz09_10min_5init_2023-06-19_00-06-46
```

19.2 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,)
```

19.3 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
```

19.4 Step 4: Specification of the Preprocessing Model

After the training and test data are specified and added to the `fun_control` dictionary, `spotPython` allows the specification of a data preprocessing pipeline, e.g., for the scaling of the data or for the one-hot encoding of categorical variables. 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})
```

19.5 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 19.5.0.1 first, before the extended implementation with `spotPython` is shown in Section 19.5.0.2.

19.5.0.1 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)
```

19.5.0.2 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
```

19.5.1 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
```

19.5.2 Comparison of the Approach Described in the PyTorch Tutorial With spotPython

Comparing the class Net from the PyTorch tutorial and the class Net_CIFAR10 from spotPython, we see that the class Net_CIFAR10 has additional attributes and does not inherit from nn directly. It adds an additional class, Net_core, that takes care of additional attributes that are common to all neural network models, e.g., the learning rate multiplier lr_mult or the batch size batch_size.

spotPython's core_model implements an instance of the Net_CIFAR10 class. In addition to the basic neural network model, the core_model can use these additional attributes. spotPython

provides methods for handling these additional attributes to guarantee 100% compatibility with the `PyTorch` classes. The method `add_core_model_to_fun_control` adds the hyperparameters and additional attributes to the `fun_control` dictionary. The method is shown below.

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

19.5.3 The Search Space: Hyperparameters

In Section 19.5.4, 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.

19.5.4 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.

19.5.5 Configuring the Search Space With spotPython

19.5.5.1 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}}
```

19.6 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.

19.6.0.1 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.

19.6.0.2 Modify Hyperparameters of Type numeric and integer (boolean)

Since the hyperparameter `k_folds` is not used in the `PyTorch` tutorial, it is de-activated here by setting the lower and upper bound to the same value. Note, `k_folds` is of type "integer".

```
from spotPython.hyperparameters.values import modify_hyper_parameter_bounds
fun_control = modify_hyper_parameter_bounds(fun_control,
    "batch_size", bounds=[1, 5])
fun_control = modify_hyper_parameter_bounds(fun_control,
    "k_folds", bounds=[0, 0])
fun_control = modify_hyper_parameter_bounds(fun_control,
    "patience", bounds=[3, 3])
```

19.6.0.3 Modify Hyperparameter of Type factor

In a similar manner as for the numerical hyperparameters, the categorical hyperparameters can be modified. New configurations can be chosen by adding or deleting levels. For example, the hyperparameter `optimizer` can be re-configured as follows:

In the following setting, two optimizers (`"SGD"` and `"Adam"`) will be compared during the `spotPython` hyperparameter tuning. The hyperparameter `optimizer` is active.

```
from spotPython.hyperparameters.values import modify_hyper_parameter_levels
fun_control = modify_hyper_parameter_levels(fun_control,
     "optimizer", ["SGD", "Adam"])
```

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

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

As discussed in Section 19.6.1, there are some issues with the LBFGS optimizer. Therefore, the usage of the LBFGS optimizer is not deactivated in `spotPython` by default. However, the LBFGS optimizer can be activated by adding it to the list of optimizers. `Rprop` was removed, because it does perform very poorly (as some pre-tests have shown). However, it can also be activated by adding it to the list of optimizers. Since `SparseAdam` does not support dense gradients, `Adam` was used instead. Therefore, there are 10 default optimizers:

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

19.6.1 Optimizers

Table 19.1 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 19.1: Optimizers available in PyTorch (selection). The default values are shown in the table.

Optimizer	lr	m	w_d	d	n	r	l_s	l_d	b	l	a	m_d	e	s_s
Adadelta	-	-	0.	-	-	0.9	1.	-	-	-	-	-	-	-
Adagrad	1e-2	-	0.	-	-	-	-	0.	-	-	-	-	-	-
Adam	1e-3	-	0.	-	-	-	-	-	a	-	-	-	-	-
AdamW	1e-3	-	1e-2	-	-	-	-	-	a	-	-	-	-	-
SparseAdam	1e-3	-	-	-	-	-	-	-	a	-	-	-	-	-
Adamax	2e-3	-	0.	-	-	-	-	-	a	-	-	-	-	-

Optimizer	lr	m	w_d	d	n	r	l_s	l_d	b	l	a	m_d	e	s_s
ASGD	1e-2	.9	0.	-	F	-	-	-	-	1e-4	.75	-	-	-
LBFGS	1.	-	-	-	-	-	-	-	-	-	-	-	-	-
NAdam	2e-3	-	0.	-	-	-	-	-	*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`.

```
fun_control = modify_hyper_parameter_bounds(fun_control,
    "lr_mult", bounds=[1.0, 1.0])
fun_control = modify_hyper_parameter_bounds(fun_control,
    "sgd_momentum", bounds=[0.9, 0.9])
```

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

19.7.1 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.

19.7.2 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).

19.7.3 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.

19.7.4 Overview of the Evaluation Settings

19.7.4.1 Settings for the Hyperparameter Tuning

An overview of the training evaluations is shown in Table 19.2. `"train_cv"` and `"test_cv"` use `sklearn.model_selection.KFold()` internally. More details on the data splitting are provided in Section 26.14 (in the Appendix).

Table 19.2: 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.

19.7.4.2 Settings for the Final Evaluation of the Tuned Architecture

19.7.4.2.1 Training of the Tuned Architecture

train_tuned(model, train): train the model with the best hyperparameter configuration (or simply the default) on the training data set. It splits the traindata into new train and validation sets using create_train_val_data_loaders(), which calls torch.utils.data.random_split() internally. Currently, 60% of the data is used for training and 40% for validation. The train data is used for training the model with train_hold_out(). The validation data is used for early stopping using validate_fold_or_hold_out() on the validation data set.

19.7.4.2.2 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 26.14.1.4 describes the final evaluation of the tuned architecture.

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

19.7.5 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})
```

19.8 Step 8: Calling the SPOT Function

19.8.1 Preparing the SPOT Call

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

```
from spotPython.hyperparameters.values import (
    get_var_type,
    get_var_name,
    get_bound_values
    )
var_type = get_var_type(fun_control)
var_name = get_var_name(fun_control)
fun_control.update({"var_type": var_type,
                    "var_name": var_name})

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

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

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

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

This allows to check if all information is available and if the information is correct. **?@tbl-design** shows the experimental design for the hyperparameter tuning. The table shows the

hyperparameters, their types, default values, lower and upper bounds, and the transformation function. The transformation function is used to transform the hyperparameter values from the unit hypercube to the original domain. The transformation function is applied to the hyperparameter values before the evaluation of the objective function. Hyperparameter transformations are shown in the column "transform", e.g., the `l1` default is `5`, which results in the value $2^5 = 32$ for the network, because the transformation `transform_power_2_int` was selected in the `JSON` file. The default value of the `batch_size` is set to `4`, which results in a batch size of $2^4 = 16$.

19.8.2 The Objective Function `fun_torch`

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

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

19.8.3 Using Default Hyperparameters or Results from Previous Runs

We add the default setting to the initial design:

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

19.8.4 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 19.9.

```
from spotPython.spot import spot
from math import inf
import numpy as np
spot_tuner = spot.Spot(fun=fun,
                    lower = lower,
```

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

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

MulticlassAccuracy: 0.4101000130176544 | Loss: 1.6304120449066162 | Acc: 0.4101000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.4731500148773193 | Loss: 1.4535225247383117 | Acc: 0.4731500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.4983499944210052 | Loss: 1.3879684734344482 | Acc: 0.4983500000000000.
Epoch: 4 |
```

```
MulticlassAccuracy: 0.5253999829292297 | Loss: 1.3114733338356017 | Acc: 0.5254000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5317000150680542 | Loss: 1.2998319584846496 | Acc: 0.5317000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5365499854087830 | Loss: 1.2773452651977539 | Acc: 0.5365500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5382500290870667 | Loss: 1.2768458868980408 | Acc: 0.5382500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5568000078201294 | Loss: 1.2427912346839904 | Acc: 0.5568000000000000.
Epoch: 9 |

MulticlassAccuracy: 0.5530999898910522 | Loss: 1.2746534210205078 | Acc: 0.5531000000000000.
Epoch: 10 |

MulticlassAccuracy: 0.5648999810218811 | Loss: 1.2362445865631104 | Acc: 0.5649000000000000.
Epoch: 11 |

MulticlassAccuracy: 0.5752500295639038 | Loss: 1.2252376949310302 | Acc: 0.5752500000000000.
Epoch: 12 |

MulticlassAccuracy: 0.5784999728202820 | Loss: 1.2062684302330018 | Acc: 0.5785000000000000.
Epoch: 13 |

MulticlassAccuracy: 0.5797500014305115 | Loss: 1.2163981791496277 | Acc: 0.5797500000000000.
Epoch: 14 |

MulticlassAccuracy: 0.5777500271797180 | Loss: 1.2217736952781677 | Acc: 0.5777500000000000.
Epoch: 15 |

MulticlassAccuracy: 0.5824999809265137 | Loss: 1.2319565966606141 | Acc: 0.5825000000000000.
Early stopping at epoch 14
Returned to Spot: Validation loss: 1.231956596660614

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

```
MulticlassAccuracy: 0.4178000092506409 | Loss: 1.6216108562469482 | Acc: 0.4178000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.4995500147342682 | Loss: 1.3795557044982911 | Acc: 0.4995500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.4955999851226807 | Loss: 1.3662989647388459 | Acc: 0.4956000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5202999711036682 | Loss: 1.3545850614130497 | Acc: 0.5203000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5376999974250793 | Loss: 1.3138996888041496 | Acc: 0.5377000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5346999764442444 | Loss: 1.3086939904332160 | Acc: 0.5347000000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5351499915122986 | Loss: 1.3008923039138318 | Acc: 0.5351500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5443999767303467 | Loss: 1.2781054498910904 | Acc: 0.5444000000000000.
Returned to Spot: Validation loss: 1.2781054498910904

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

MulticlassAccuracy: 0.1001999974250793 | Loss: 2.3095021947860719 | Acc: 0.1002000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.0971499979496002 | Loss: 2.3064508244752884 | Acc: 0.0971500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.0995500013232231 | Loss: 2.3071898629426957 | Acc: 0.0995500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.0995500013232231 | Loss: 2.3055451205730439 | Acc: 0.0995500000000000.
Epoch: 5 |
```

```
MulticlassAccuracy: 0.0995500013232231 | Loss: 2.3099301098585130 | Acc: 0.0995500000000000.
Epoch: 6 |

MulticlassAccuracy: 0.0971499979496002 | Loss: 2.3056820832729339 | Acc: 0.0971500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.0980999991297722 | Loss: 2.3066921489953995 | Acc: 0.0981000000000000.
Early stopping at epoch 6
Returned to Spot: Validation loss: 2.3066921489953995

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

MulticlassAccuracy: 0.3942500054836273 | Loss: 1.6123133260846139 | Acc: 0.3942500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.4570499956607819 | Loss: 1.4608794206291438 | Acc: 0.4570500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.4983499944210052 | Loss: 1.3746368969544769 | Acc: 0.4983500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5269500017166138 | Loss: 1.3196217773199082 | Acc: 0.5269500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5333999991416931 | Loss: 1.3144403619609772 | Acc: 0.5334000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5535500049591064 | Loss: 1.2666505189999937 | Acc: 0.5535500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5601500272750854 | Loss: 1.2723937103066594 | Acc: 0.5601500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5662999749183655 | Loss: 1.2596585603520274 | Acc: 0.5663000000000000.
Returned to Spot: Validation loss: 1.2596585603520274

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

```
MulticlassAccuracy: 0.4390999972820282 | Loss: 1.5167334157466887 | Acc: 0.4391000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.4677500128746033 | Loss: 1.4511797853469848 | Acc: 0.4677500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.4848999977111816 | Loss: 1.4082916311740876 | Acc: 0.4849000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5059000253677368 | Loss: 1.3631655582189559 | Acc: 0.5059000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5140500068664551 | Loss: 1.3540844504833220 | Acc: 0.5140500000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5267000198364258 | Loss: 1.3128073700904845 | Acc: 0.5266999999999999.
Epoch: 7 |

MulticlassAccuracy: 0.5393999814987183 | Loss: 1.2878062868356706 | Acc: 0.5394000000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5411499738693237 | Loss: 1.2821812673091888 | Acc: 0.5411500000000000.
Epoch: 9 |

MulticlassAccuracy: 0.5493999719619751 | Loss: 1.2597104520320892 | Acc: 0.5494000000000000.
Epoch: 10 |

MulticlassAccuracy: 0.5544999837875366 | Loss: 1.2486023411035538 | Acc: 0.5545000000000000.
Epoch: 11 |

MulticlassAccuracy: 0.5577499866485596 | Loss: 1.2415274378776551 | Acc: 0.5577500000000000.
Epoch: 12 |

MulticlassAccuracy: 0.5627499818801880 | Loss: 1.2376774848461152 | Acc: 0.5627500000000000.
Epoch: 13 |

MulticlassAccuracy: 0.5643500089645386 | Loss: 1.2261771851062775 | Acc: 0.5643500000000000.
Epoch: 14 |
```

```
MulticlassAccuracy: 0.5652499794960022 | Loss: 1.2242570672273636 | Acc: 0.5652500000000000.
Epoch: 15 |

MulticlassAccuracy: 0.5671499967575073 | Loss: 1.2231163112878800 | Acc: 0.5671500000000000.
Epoch: 16 |

MulticlassAccuracy: 0.5698500275611877 | Loss: 1.2206852425098420 | Acc: 0.5698500000000000.
Returned to Spot: Validation loss: 1.220685242509842

config: {'l1': 64, 'l2': 128, 'lr_mult': 1.0, 'batch_size': 16, 'epochs': 16, 'k_folds': 0,
Epoch: 1 |

MulticlassAccuracy: 0.4535999894142151 | Loss: 1.5110390196323396 | Acc: 0.4536000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.4830000102519989 | Loss: 1.4341976581573486 | Acc: 0.4830000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.5011500120162964 | Loss: 1.3874174671649933 | Acc: 0.5011500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5160499811172485 | Loss: 1.3530661789417266 | Acc: 0.5160500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5205000042915344 | Loss: 1.3340329530239106 | Acc: 0.5205000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5245000123977661 | Loss: 1.3314349993705750 | Acc: 0.5245000000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5349000096321106 | Loss: 1.3002712713718414 | Acc: 0.5349000000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5404000282287598 | Loss: 1.2899356038093568 | Acc: 0.5404000000000000.
Epoch: 9 |

MulticlassAccuracy: 0.5463500022888184 | Loss: 1.2788521531581878 | Acc: 0.5463500000000000.
Epoch: 10 |
```

```
MulticlassAccuracy: 0.5504000186920166 | Loss: 1.2752792395114898 | Acc: 0.5504000000000000.
Epoch: 11 |

MulticlassAccuracy: 0.5543000102043152 | Loss: 1.2551528417110442 | Acc: 0.5543000000000000.
Epoch: 12 |

MulticlassAccuracy: 0.5537499785423279 | Loss: 1.2538598964691161 | Acc: 0.5537500000000000.
Epoch: 13 |

MulticlassAccuracy: 0.5566499829292297 | Loss: 1.2477908047199249 | Acc: 0.5566500000000000.
Epoch: 14 |

MulticlassAccuracy: 0.5577999949455261 | Loss: 1.2468155496120452 | Acc: 0.5578000000000000.
Epoch: 15 |

MulticlassAccuracy: 0.5645499825477600 | Loss: 1.2320502506017685 | Acc: 0.5645500000000000.
Epoch: 16 |

MulticlassAccuracy: 0.5644500255584717 | Loss: 1.2317526315212250 | Acc: 0.5644500000000000.
Returned to Spot: Validation loss: 1.231752631521225

spotPython tuning: 1.220685242509842 [###-------] 33.88%


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

MulticlassAccuracy: 0.4891499876976013 | Loss: 1.4665832023153198 | Acc: 0.4891500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.5249999761581421 | Loss: 1.4087849099246130 | Acc: 0.5250000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.5163499712944031 | Loss: 1.5556861498441377 | Acc: 0.5163500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5498499870300293 | Loss: 1.3916781537996286 | Acc: 0.5498499999999999.
Epoch: 5 |
```

```
MulticlassAccuracy: 0.5494499802589417 | Loss: 1.5300120123093512 | Acc: 0.5494500000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5569499731063843 | Loss: 1.4943019928745995 | Acc: 0.5569499999999999.
Epoch: 7 |

MulticlassAccuracy: 0.5763499736785889 | Loss: 1.4172966401714968 | Acc: 0.5763500000000000.
Early stopping at epoch 6
Returned to Spot: Validation loss: 1.4172966401714968

spotPython tuning: 1.220685242509842 [######----] 59.35%


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

MulticlassAccuracy: 0.4937500059604645 | Loss: 1.3980224881649017 | Acc: 0.4937500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.5206000208854675 | Loss: 1.3439587865829468 | Acc: 0.5206000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.5622000098228455 | Loss: 1.2250800273418427 | Acc: 0.5622000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5756999850273132 | Loss: 1.1895112948894502 | Acc: 0.5757000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5857499837875366 | Loss: 1.1810582669258118 | Acc: 0.5857500000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5863999724388123 | Loss: 1.1676061150312425 | Acc: 0.5864000000000000.
Epoch: 7 |

MulticlassAccuracy: 0.6011000275611877 | Loss: 1.1610746207714080 | Acc: 0.6011000000000000.
Epoch: 8 |

MulticlassAccuracy: 0.6010000109672546 | Loss: 1.1491030423402786 | Acc: 0.6010000000000000.
Epoch: 9 |
```

```
MulticlassAccuracy: 0.6021500229835510 | Loss: 1.1679993093967438 | Acc: 0.6021500000000000.
Epoch: 10 |

MulticlassAccuracy: 0.5965999960899353 | Loss: 1.1902096559286117 | Acc: 0.5966000000000000.
Epoch: 11 |

MulticlassAccuracy: 0.6068500280380249 | Loss: 1.2108325213789939 | Acc: 0.6068500000000000.
Early stopping at epoch 10
Returned to Spot: Validation loss: 1.2108325213789939

spotPython tuning: 1.2108325213789939 [########--] 83.25%


config: {'l1': 64, 'l2': 512, 'lr_mult': 1.0, 'batch_size': 32, 'epochs': 16, 'k_folds': 0,
Epoch: 1 |

MulticlassAccuracy: 0.4492999911308289 | Loss: 1.4874671487808226 | Acc: 0.4493000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.5135999917984009 | Loss: 1.3465182330131531 | Acc: 0.5135999999999999.
Epoch: 3 |

MulticlassAccuracy: 0.5418000221252441 | Loss: 1.2865717721939087 | Acc: 0.5417999999999999.
Epoch: 4 |

MulticlassAccuracy: 0.5518000125885010 | Loss: 1.2430488024711608 | Acc: 0.5518000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5745000243186951 | Loss: 1.2057166361808778 | Acc: 0.5745000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5886499881744385 | Loss: 1.1529680906295776 | Acc: 0.5886500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5787000060081482 | Loss: 1.2009244564056396 | Acc: 0.5787000000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5914499759674072 | Loss: 1.1621136879920959 | Acc: 0.5914500000000000.
Epoch: 9 |
```

```
MulticlassAccuracy: 0.5842499732971191 | Loss: 1.2132302490234375 | Acc: 0.5842500000000000.
Early stopping at epoch 8
Returned to Spot: Validation loss: 1.2132302490234375

spotPython tuning: 1.2108325213789939 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x2b1a56bf0>
```

19.9 Step 9: Tensorboard

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

19.9.1 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.

19.9.2 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:
```

Figure 19.1: 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 19.2: Tensorboard

```
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)
```

19.10 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 19.3: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

?@fig-progress shows a typical behaviour that can be observed in many hyperparameter studies (Bartz et al. 2022): the largest improvement is obtained during the evaluation of the initial design. The surrogate model based optimization-optimization with the surrogate refines the results. **?@fig-progress** also illustrates one major difference between `ray[tune]` as used in PyTorch (2023a) and `spotPython`: the `ray[tune]` uses a random search and will generate results similar to the *black* dots, whereas `spotPython` uses a surrogate model based optimization and presents results represented by *red* dots in **?@fig-progress**. The surrogate model based optimization is considered to be more efficient than a random search, because the surrogate model guides the search towards promising regions in the hyperparameter space.

In addition to the improved ("optimized") hyperparameter values, `spotPython` allows a statistical analysis, e.g., a sensitivity analysis, of the results. We can print the results of the hyperparameter tuning, see **?@tbl-results**. The table shows the hyperparameters, their types, default

values, lower and upper bounds, and the transformation function. The column "tuned" shows the tuned values. The column "importance" shows the importance of the hyperparameters. The column "stars" shows the importance of the hyperparameters in stars. The importance is computed by the SPOT software.

```
from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control=fun_control, spot=spot_tuner))
```

```
| name         | type   | default   |   lower |   upper |   tuned | transform              |
|--------------|--------|-----------|---------|---------|---------|------------------------|--
| l1           | int    | 5         |     2.0 |     9.0 |     6.0 | transform_power_2_int  |
| l2           | int    | 5         |     2.0 |     9.0 |     8.0 | transform_power_2_int  |
| lr_mult      | float  | 1.0       |     1.0 |     1.0 |     1.0 | None                   |
| batch_size   | int    | 4         |     1.0 |     5.0 |     4.0 | transform_power_2_int  |
| epochs       | int    | 3         |     3.0 |     4.0 |     4.0 | transform_power_2_int  |
| k_folds      | int    | 1         |     0.0 |     0.0 |     0.0 | None                   |
| patience     | int    | 5         |     3.0 |     3.0 |     3.0 | None                   |
| optimizer    | factor | SGD       |     0.0 |     9.0 |     2.0 | 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 19.4: Variable importance plot, threshold 0.025.

19.10.1 Get the Tuned Architecture (SPOT Results)

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

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

```
Net_CIFAR10(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=10, bias=True)
)
```

19.10.2 Get Default Hyperparameters

In a similar manner as in **?@sec-get-spot-results**, the default hyperparameters can be obtained.

```
# fun_control was modified, we generate a new one with the original
# default hyperparameters
from spotPython.hyperparameters.values import get_one_core_model_from_X
fc = fun_control
fc.update({"core_model_hyper_dict":
    hyper_dict[fun_control["core_model"].__name__]})
model_default = get_one_core_model_from_X(X_start, fun_control=fc)
model_default
```

```
Net_CIFAR10(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=32, bias=True)
  (fc2): Linear(in_features=32, out_features=32, bias=True)
  (fc3): Linear(in_features=32, out_features=10, bias=True)
)
```

19.10.3 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: 0.1038499996066093 | Loss: 2.3025718597412110 | Acc: 0.1038500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.1056500002741814 | Loss: 2.3001274030685424 | Acc: 0.1056500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.1238000020384789 | Loss: 2.2965125730514528 | Acc: 0.1238000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.1533499956130981 | Loss: 2.2894232133865358 | Acc: 0.1533500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.1835000067949295 | Loss: 2.2737157850265501 | Acc: 0.1835000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.1767500042915344 | Loss: 2.2274773530960084 | Acc: 0.1767500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.2153500020503998 | Loss: 2.1469441737174986 | Acc: 0.2153500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.2574999928474426 | Loss: 2.0730154965400698 | Acc: 0.2575000000000000.
Returned to Spot: Validation loss: 2.07301549654007

MulticlassAccuracy: 0.2628999948501587 | Loss: 2.0697322952270509 | Acc: 0.2629000000000000.
Final evaluation: Validation loss: 2.069732295227051
Final evaluation: Validation metric: 0.2628999948501587
---------------------------------------------
```

```
(2.069732295227051, nan, tensor(0.2629))
```

19.10.4 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: 0.4666500091552734 | Loss: 1.4695183675765990 | Acc: 0.4666500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.5285999774932861 | Loss: 1.3158676015853883 | Acc: 0.5286000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.5346000194549561 | Loss: 1.2812264657020569 | Acc: 0.5346000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5705999732017517 | Loss: 1.2019315437555314 | Acc: 0.5706000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5936999917030334 | Loss: 1.1425811416864395 | Acc: 0.5937000000000000.
Epoch: 6 |
```

```
MulticlassAccuracy: 0.5740000009536743 | Loss: 1.2210487580299378 | Acc: 0.5740000000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5995000004768372 | Loss: 1.1590382073163987 | Acc: 0.5995000000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5914499759674072 | Loss: 1.1887884808301925 | Acc: 0.5914500000000000.
Early stopping at epoch 7
Returned to Spot: Validation loss: 1.1887884808301925

MulticlassAccuracy: 0.5795000195503235 | Loss: 1.2003105345726013 | Acc: 0.5795000000000000.
Final evaluation: Validation loss: 1.2003105345726013
Final evaluation: Validation metric: 0.5795000195503235
----------------------------------------------

(1.2003105345726013, nan, tensor(0.5795))
```

19.10.5 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:  15.75142451441451
optimizer:  100.0
```

Figure 19.5: Contour plots.

The figures (**?@fig-contour**) show the contour plots of the loss as a function of the hyperparameters. These plots are very helpful for benchmark studies and for understanding neural networks. `spotPython` provides additional tools for a visual inspection of the results and give valuable insights into the hyperparameter tuning process. This is especially useful for model explainability, transparency, and trustworthiness. In addition to the contour plots, **?@fig-parallel** shows the parallel plot of the hyperparameters.

```
spot_tuner.parallel_plot()
```

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

Parallel coordinates plots

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

19.11 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.

19.12 Appendix

19.12.1 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
```

20 HPT: sklearn RandomForestClassifier VBDP Data

This document refers to the following software versions:

- `python`: 3.10.10

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

```
spotPython                  0.2.38
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
```

20.1 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_bartz09_1min_5init_2023-06-19_00-33-58
```

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

20.2 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")
```

20.3 Step 3: PyTorch Data Loading

20.3.1 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`.

20.3.2 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})
```

20.4 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(),
#     )
```

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

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

`fun_control = add_core_model_to_fun_control(SVC, fun_control, SklearnHyperDict)`

Other core_models are, e.g.,:

- RidgeCV
- GradientBoostingRegressor
- ElasticNet
- RandomForestClassifier
- LogisticRegression
- KNeighborsClassifier
- RandomForestClassifier
- GradientBoostingClassifier
- HistGradientBoostingClassifier

We will use the `RandomForestClassifier` classifier in this example.

```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
fun_control = add_core_model_to_fun_control(core_model=core_model,
                              fun_control=fun_control,
                              hyper_dict=SklearnHyperDict,
                              filename=None)
```

Now `fun_control` has the information from the JSON file. The available hyperparameters are:

```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
```

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

20.6.1 Modify hyperparameter of type numeric and integer (boolean)

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

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

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

20.6.2 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 19.6.

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

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

The new setting can be controlled via:

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

```
from spotPython.hyperparameters.values import modify_hyper_parameter_levels
# XGBoost:
# fun_control = modify_hyper_parameter_levels(fun_control, "loss", ["log_loss"])
```

> **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 20.7.3.

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

20.6.3 Optimizers

Optimizers are described in Section 19.6.1.

20.6.4 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.

20.7 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"`.

20.7.1 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.

20.7.1.1 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}`.

20.7.1.2 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},
               })
```

20.7.2 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",
})
```

20.7.3 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",
})
fun_control = modify_hyper_parameter_bounds(fun_control, "bootstrap", bounds=[1, 1])
fun_control = modify_hyper_parameter_bounds(fun_control, "oob_score", bounds=[1, 1])
```

20.7.3.1 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,
# })
```

20.8 Step 8: Calling the SPOT Function

20.8.1 Preparing the SPOT Call

- Get types and variable names as well as lower and upper bounds for the hyperparameters.

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

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

```
| name                      | type   | default   |   lower |   upper | transform
|---------------------------|--------|-----------|---------|---------|-----------------------
| n_estimators              | int    | 7         |       5 |   10    | transform_power_2_int
| criterion                 | factor | gini      |       0 |    2    | None
| max_depth                 | int    | 10        |       1 |   20    | transform_power_2_int
| min_samples_split         | int    | 2         |       2 |  100    | None
| min_samples_leaf          | int    | 1         |       1 |   25    | None
| min_weight_fraction_leaf  | float  | 0.0       |       0 |    0.01 | None
| max_features              | factor | sqrt      |       0 |    1    | transform_none_to_None
| max_leaf_nodes            | int    | 10        |       7 |   12    | transform_power_2_int
| min_impurity_decrease     | float  | 0.0       |       0 |    0.01 | None
| bootstrap                 | factor | 1         |       1 |    1    | None
| oob_score                 | factor | 0         |       1 |    1    | None
```

20.8.2 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
```

20.8.3 Run the Spot Optimizer

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

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

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

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

```
                                        "min_theta": -4,
                                        "max_theta": 3,
                                        "n_theta": len(var_name),
                                        "model_fun_evals": 10_000,
                                        "log_level": 50
                                        })
spot_tuner.run(X_start=X_start)
```

```
spotPython tuning: -0.3474842767295597 [----------] 0.64%

spotPython tuning: -0.35251572327044023 [----------] 2.64%

spotPython tuning: -0.3559748427672956 [----------] 4.89%

spotPython tuning: -0.3559748427672956 [#---------] 6.75%

spotPython tuning: -0.3559748427672956 [#---------] 9.40%

spotPython tuning: -0.36194968553459117 [#---------] 11.25%

spotPython tuning: -0.36194968553459117 [#---------] 13.46%

spotPython tuning: -0.36194968553459117 [##--------] 15.18%

spotPython tuning: -0.36194968553459117 [##--------] 16.33%

spotPython tuning: -0.36194968553459117 [##--------] 18.58%

spotPython tuning: -0.36289308176100626 [##--------] 20.92%

spotPython tuning: -0.36289308176100626 [##--------] 23.07%

spotPython tuning: -0.36289308176100626 [###-------] 26.65%

spotPython tuning: -0.36289308176100626 [###-------] 30.97%

spotPython tuning: -0.36289308176100626 [####------] 35.27%
```

```
spotPython tuning: -0.36289308176100626 [####------] 39.54%

spotPython tuning: -0.36289308176100626 [####------] 43.42%

spotPython tuning: -0.36383647798742136 [#####-----] 47.88%

spotPython tuning: -0.36383647798742136 [#####-----] 51.45%

spotPython tuning: -0.36383647798742136 [######----] 56.86%

spotPython tuning: -0.36383647798742136 [######----] 61.49%

spotPython tuning: -0.3660377358490566 [#######---] 65.91%

spotPython tuning: -0.3660377358490566 [#######---] 70.35%

spotPython tuning: -0.3660377358490566 [#######---] 74.30%

spotPython tuning: -0.3660377358490566 [########--] 78.17%

spotPython tuning: -0.3663522012578616 [########--] 82.51%

spotPython tuning: -0.3663522012578616 [#########-] 86.35%

spotPython tuning: -0.3663522012578616 [#########-] 90.60%

spotPython tuning: -0.3663522012578616 [##########] 96.70%

spotPython tuning: -0.3663522012578616 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x16b07ac50>
```

20.9 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section 19.9, see also the description in the documentation: Tensorboard.

20.10 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 20.1: 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	10.0
criterion	factor	gini	0.0	2.0	1.0
max_depth	int	10	1.0	20.0	15.0
min_samples_split	int	2	2.0	100.0	12.0
min_samples_leaf	int	1	1.0	25.0	1.0
min_weight_fraction_leaf	float	0.0	0.0	0.01	0.005529334422043133
max_features	factor	sqrt	0.0	1.0	0.0
max_leaf_nodes	int	10	7.0	12.0	10.0
min_impurity_decrease	float	0.0	0.0	0.01	0.0023984865182785976
bootstrap	factor	1	1.0	1.0	1.0
oob_score	factor	0	1.0	1.0	1.0

20.10.1 Show variable importance

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

Figure 20.2: Variable importance plot, threshold 0.025.

20.10.2 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))])
```

20.10.3 Get SPOT Results

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

```
[[1.00000000e+01 1.00000000e+00 1.50000000e+01 1.20000000e+01
  1.00000000e+00 5.52933442e-03 0.00000000e+00 1.00000000e+01
  2.39848652e-03 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': 1024,
  'criterion': 'entropy',
  'max_depth': 32768,
  'min_samples_split': 12,
  'min_samples_leaf': 1,
  'min_weight_fraction_leaf': 0.005529334422043133,
  'max_features': 'sqrt',
  'max_leaf_nodes': 1024,
  'min_impurity_decrease': 0.0023984865182785976,
  'bootstrap': 1,
  'oob_score': 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
```

```
RandomForestClassifier(bootstrap=1, criterion='entropy', max_depth=32768,
                       max_leaf_nodes=1024,
                       min_impurity_decrease=0.0023984865182785976,
                       min_samples_split=12,
                       min_weight_fraction_leaf=0.005529334422043133,
                       n_estimators=1024, oob_score=1)
```

20.10.4 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
```

```
0.3559322033898305
```

```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
```

20.10.5 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: 0.36186440677966103
std_res: 0.006596491572601693
min_res: 0.3455743879472693
max_res: 0.3757062146892655
median_res: 0.36205273069679844
```

20.10.6 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)
```

```
0.3352165725047081
```

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: 0.34318895166352786
std_res: 0.01475017017438103
min_res: 0.31544256120527303
max_res: 0.3681732580037665
median_res: 0.34133709981167604
```

20.10.7 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)
```

```
(-0.3663522012578616, -0.2921383647798742)
```

20.10.8 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)
```

```
(0.36603773584905663, None)
```

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

```
(0.32614379084967315, 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)
```

```
(0.3669885982562039, None)
```

20.10.9 Detailed Hyperparameter Plots

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

```
max_depth:  0.08291092578443698
min_samples_split:  0.38395434675213963
min_samples_leaf:  0.10215818136941661
max_features:  100.0
```

Surrogate
x3: min_samples_split
x2: max_depth
−0.296
−0.304
−0.312
−0.320
−0.328
−0.336
−0.30
−0.31
−0.32
−0.33
x2: max_depth
x3: min_samples_split
Surrogate
x4: min_samples_leaf
x2: max_depth
−0.300
−0.305
−0.310
−0.315
−0.320
−0.325
−0.330
−0.31
−0.32
x2: max_depth
x4: min_samples_leaf
Surrogate
x6: max_features
x2: max_depth
−0.3104
−0.3120
−0.3136
−0.3152
−0.3168
−0.3184
−0.312
−0.314
−0.316
−0.318
x2: max_depth
x6: max_features

Surrogate
x4: min_samples_leaf
x3: min_samples_split
−0.295
−0.305
−0.315
−0.325
−0.335
−0.345
−0.355
x3: min_samples_split
x4: min_samples_leaf
−0.30
−0.32
−0.34

Surrogate
x6: max_features
x3: min_samples_split
−0.296
−0.304
−0.312
−0.320
−0.328
−0.336
x3: min_samples_split
x6: max_features
−0.30
−0.31
−0.32
−0.33

Surrogate
x6: max_features
x4: min_samples_leaf
−0.306
−0.310
−0.314
−0.318
−0.322
−0.326
x4: min_samples_leaf
x6: max_features
−0.310
−0.315
−0.320
−0.325

20.10.10 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
```

20.10.11 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)
```

21 HPT: sklearn XGB Classifier VBDP Data

This document refers to the following software versions:

- python: 3.10.10

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

```
spotPython                  0.2.38
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
```

21.1 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_bartz09_1min_5init_2023-06-19_00-45-12
```

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

21.2 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")
```

21.3 Step 3: PyTorch Data Loading

21.3.1 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`.

21.3.2 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})
```

21.4 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(),
#     )
```

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

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

`fun_control = add_core_model_to_fun_control(SVC, fun_control, SklearnHyperDict)`

Other core_models are, e.g.,:

- RidgeCV
- GradientBoostingRegressor
- ElasticNet
- RandomForestClassifier
- LogisticRegression
- KNeighborsClassifier
- RandomForestClassifier
- GradientBoostingClassifier
- HistGradientBoostingClassifier

We will use the `RandomForestClassifier` classifier in this example.

```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
fun_control = add_core_model_to_fun_control(core_model=core_model,
                              fun_control=fun_control,
                              hyper_dict=SklearnHyperDict,
                              filename=None)
```

Now `fun_control` has the information from the JSON file. The available hyperparameters are:

```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
```

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

21.6.1 Modify hyperparameter of type numeric and integer (boolean)

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

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

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

21.6.2 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 19.6.

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

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

The new setting can be controlled via:

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

```python
from spotPython.hyperparameters.values import modify_hyper_parameter_levels
# XGBoost:
fun_control = modify_hyper_parameter_levels(fun_control, "loss", ["log_loss"])
```

21.6.3 Optimizers

Optimizers are described in Section 19.6.1.

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

21.7.1 Evaluation

The evaluation procedure requires the specification of two elements:

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

21.7.2 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.

21.7.3 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"`.

21.7.4 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.

21.7.4.1 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}`.

21.7.4.2 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},
               })
```

21.7.5 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",
})
```

21.7.5.1 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,
# })
```

21.8 Step 8: Calling the SPOT Function

21.8.1 Preparing the SPOT Call

- Get types and variable names as well as lower and upper bounds for the hyperparameters.

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

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

```
| name              | type   | default   |   lower |   upper | transform             |
|-------------------|--------|-----------|---------|---------|-----------------------|
| loss              | factor | log_loss  |   0     |   0     | None                  |
| learning_rate     | float  | -1.0      |  -5     |   0     | transform_power_10    |
| max_iter          | int    | 7         |   3     |  10     | transform_power_2_int |
| max_leaf_nodes    | int    | 5         |   1     |  12     | transform_power_2_int |
| max_depth         | int    | 2         |   1     |  20     | transform_power_2_int |
| min_samples_leaf  | int    | 4         |   2     |  10     | transform_power_2_int |
| l2_regularization | float  | 0.0       |   0     |  10     | None                  |
| max_bins          | int    | 255       | 127     | 255     | None                  |
| early_stopping    | factor | 1         |   0     |   1     | None                  |
| n_iter_no_change  | int    | 10        |   5     |  20     | None                  |
| tol               | float  | 0.0001    |   1e-05 |   0.001 | None                  |
```

21.8.2 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
```

21.8.3 Run the `Spot` Optimizer

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

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

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

```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: -0.3458646616541353 [----------] 3.80%

spotPython tuning: -0.3458646616541353 [#---------] 5.96%

spotPython tuning: -0.3458646616541353 [#---------] 7.43%

spotPython tuning: -0.3458646616541353 [##--------] 22.61%

spotPython tuning: -0.35714285714285715 [###-------] 26.73%
```

```
spotPython tuning: -0.35714285714285715 [###-------] 30.45%

spotPython tuning: -0.35714285714285715 [###-------] 32.53%

spotPython tuning: -0.35714285714285715 [####------] 35.34%

spotPython tuning: -0.35714285714285715 [####------] 36.99%

spotPython tuning: -0.3596491228070175 [####------] 41.62%

spotPython tuning: -0.3596491228070175 [#####-----] 46.12%

spotPython tuning: -0.3596491228070175 [#####-----] 51.87%

spotPython tuning: -0.3596491228070175 [######----] 62.02%

spotPython tuning: -0.3659147869674185 [#######---] 71.07%

spotPython tuning: -0.3746867167919799 [########--] 80.36%

spotPython tuning: -0.3746867167919799 [#########-] 89.55%

spotPython tuning: -0.3746867167919799 [##########] 96.13%

spotPython tuning: -0.3746867167919799 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x14a528100>
```

21.9 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section 19.9, see also the description in the documentation: Tensorboard.

21.10 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 21.1: 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 |   -1.3754456322392838 | trans
| max_iter           | int    | 7          |     3.0 |    10.0 |                    9.0 | trans
| max_leaf_nodes     | int    | 5          |     1.0 |    12.0 |                    1.0 | trans
| max_depth          | int    | 2          |     1.0 |    20.0 |                   10.0 | trans
| min_samples_leaf   | int    | 4          |     2.0 |    10.0 |                    4.0 | trans
| l2_regularization  | float  | 0.0        |     0.0 |    10.0 |     5.763523843455282 | None
| max_bins           | int    | 255        |   127.0 |   255.0 |                  137.0 | None
| early_stopping     | factor | 1          |     0.0 |     1.0 |                    1.0 | None
| n_iter_no_change   | int    | 10         |     5.0 |    20.0 |                   20.0 | None
| tol                | float  | 0.0001     |   1e-05 |   0.001 | 0.0005255783429491349 | None
```

21.10.1 Show variable importance

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

Figure 21.2: Variable importance plot, threshold 0.025.

21.10.2 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}
```

```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),
                ('histgradientboostingclassifier',
                 HistGradientBoostingClassifier(early_stopping=1, max_depth=4,
                                                max_iter=128, max_leaf_nodes=32,
                                                min_samples_leaf=16,
                                                tol=0.0001))])
```

21.10.3 Get SPOT Results

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

```
[[ 0.00000000e+00 -1.37544563e+00  9.00000000e+00  1.00000000e+00
   1.00000000e+01  4.00000000e+00  5.76352384e+00  1.37000000e+02
   1.00000000e+00  2.00000000e+01  5.25578343e-04]]
```

```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)
```

```
[{'loss': 'log_loss',
  'learning_rate': 0.042126401997741446,
  'max_iter': 512,
  'max_leaf_nodes': 2,
  'max_depth': 1024,
  'min_samples_leaf': 16,
  'l2_regularization': 5.763523843455282,
  'max_bins': 137,
  'early_stopping': 1,
  'n_iter_no_change': 20,
  'tol': 0.0005255783429491349}]
```

```
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=5.763523843455282,
                               learning_rate=0.042126401997741446, max_bins=137,
                               max_depth=1024, max_iter=512, max_leaf_nodes=2,
                               min_samples_leaf=16, n_iter_no_change=20,
                               tol=0.0005255783429491349)
```

21.10.4 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
```

```
0.3418079096045198
```

```
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
```

21.10.5 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: 0.3415568110483366
std_res: 0.010297528522176665
min_res: 0.32015065913371
max_res: 0.3615819209039548
median_res: 0.34133709981167604
```

21.10.6 Evalution of the Default Hyperparameters

```
model_default.fit(X_train, y_train)["histgradientboostingclassifier"]
```

```
HistGradientBoostingClassifier(early_stopping=1, max_depth=4, max_iter=128,
                               max_leaf_nodes=32, min_samples_leaf=16,
                               tol=0.0001)
```

- One evaluation of the default hyperparameters is performed on the hold-out test set.

```
y_pred = model_default.predict_proba(X_test)
mapk_score(y_true=y_test, y_pred=y_pred, k=3)
```

```
0.3352165725047081
```

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: 0.3403640929064658
std_res: 0.014450580105688456
min_res: 0.30414312617702444
max_res: 0.3804143126177025
median_res: 0.33945386064030125
```

21.10.7 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)
```

```
(-0.3746867167919799, -0.20927318295739344)
```

21.10.8 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)
```

```
(0.33522012578616345, None)
```

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

```
(0.24373638344226575, 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)
```

```
(0.3285446009389671, None)
```

21.10.9 Detailed Hyperparameter Plots

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

```
learning_rate:  91.62142872957374
max_iter:  29.24882137694548
max_leaf_nodes:  1.2571269865221915
min_samples_leaf:  100.0
max_bins:  0.4854942279795972
early_stopping:  0.20524324002483432
n_iter_no_change:  0.5672764640441417
tol:  10.933839953634553
```

Surrogate
x1: max_iter
x0: learning_rate
−0.225
−0.235
−0.245
−0.255
−0.265
−0.275
−0.23
−0.24
−0.25
−0.26
−0.27
Surrogate
x2: max_leaf_nodes
x0: learning_rate
−0.225
−0.235
−0.245
−0.255
−0.265
−0.275
−0.23
−0.24
−0.25
−0.26
−0.27
Surrogate
x4: min_samples_leaf
x0: learning_rate
−0.208
−0.224
−0.240
−0.256
−0.272
−0.288
−0.22
−0.24
−0.26
−0.28

Surrogate
x6: max_bins
x0: learning_rate
240
220
200
180
160
140
−5
−4
−3
−2
−1
0
−0.225
−0.235
−0.245
−0.255
−0.265
−0.275

x0: learning_rate
x6: max_bins
−4
−2
0
150
200
250
−0.23
−0.24
−0.25
−0.26
−0.27

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
−0.21
−0.23
−0.25
−0.27
−0.29
−0.31

x0: learning_rate
x7: early_stopping
−4
−2
0
0.00
0.25
0.50
0.75
1.00
−0.22
−0.24
−0.26
−0.28
−0.30

Surrogate
x8: n_iter_no_change
x0: learning_rate
20.0
17.5
15.0
12.5
10.0
7.5
5.0
−5
−4
−3
−2
−1
0
−0.225
−0.235
−0.245
−0.255
−0.265
−0.275

x0: learning_rate
x8: n_iter_no_change
−4
−2
0
5
10
15
20
−0.23
−0.24
−0.25
−0.26
−0.27

Surrogate
x9: tol
x0: learning_rate
0.0010
0.0008
0.0006
0.0004
0.0002
−5
−4
−3
−2
−1
0
−0.225
−0.235
−0.245
−0.255
−0.265
−0.275

x0: learning_rate
x9: tol
−4
−2
0
0.00100
0.00075
0.00050
0.00025
0.00000
−0.23
−0.24
−0.25
−0.26
−0.27

Surrogate
x2: max_leaf_nodes
x1: max_iter
12
10
8
6
4
2
4
6
8
10
−0.2504
−0.2520
−0.2536
−0.2552
−0.2568
−0.2584

x1: max_iter
x2: max_leaf_nodes
4
6
8
10
2.5
5.0
7.5
10.0
12.5
−0.252
−0.254
−0.256
−0.258

Surrogate
x4: min_samples_leaf
x1: max_iter
10
8
6
4
2
4
6
8
10
−0.210
−0.222
−0.234
−0.246
−0.258
−0.270
−0.282

x1: max_iter
x4: min_samples_leaf
4
6
8
10
2
4
6
8
10
−0.22
−0.24
−0.26

Surrogate
x6: max_bins
x1: max_iter
−0.2512
−0.2528
−0.2544
−0.2560
−0.2576
−0.2592
x1: max_iter
x6: max_bins
Surrogate
x7: early_stopping
x1: max_iter
−0.252
−0.256
−0.260
−0.264
−0.268
−0.272
x1: max_iter
x7: early_stopping
Surrogate
x8: n_iter_no_change
x1: max_iter
−0.2512
−0.2528
−0.2544
−0.2560
−0.2576
−0.2592
x1: max_iter
x8: n_iter_no_change

Surrogate
0.0010
0.0008
0.0006
0.0004
0.0002
x9: tol
4
6
8
10
x1: max_iter
−0.249
−0.251
−0.253
−0.255
−0.257
−0.259

−0.250
−0.252
−0.254
−0.256
−0.258
0.00100
0.00075
0.00050
0.00025
0.00000
x9: tol
4
6
8
10
x1: max_iter

Surrogate
10
8
6
4
2
x4: min_samples_leaf
2
4
6
8
10
12
x2: max_leaf_nodes
−0.216
−0.228
−0.240
−0.252
−0.264
−0.276

−0.22
−0.24
−0.26
2.5
5.0
7.5
10.0
12.5
x2: max_leaf_nodes
2
4
6
8
10
x4: min_samples_leaf

Surrogate
240
220
200
180
160
140
x6: max_bins
2
4
6
8
10
12
x2: max_leaf_nodes
−0.2550
−0.2555
−0.2560
−0.2565
−0.2570
−0.2575
−0.2580
−0.2585
−0.2590
−0.2595

−0.256
−0.257
−0.258
−0.259
2.5
5.0
7.5
10.0
12.5
x2: max_leaf_nodes
250
200
150
x6: max_bins

Surrogate
x7: early_stopping
x2: max_leaf_nodes
−0.254
−0.258
−0.262
−0.266
−0.270
−0.274
x2: max_leaf_nodes
x7: early_stopping

Surrogate
x8: n_iter_no_change
x2: max_leaf_nodes
−0.2545
−0.2555
−0.2565
−0.2575
−0.2585
−0.2595
x2: max_leaf_nodes
x8: n_iter_no_change

Surrogate
x9: tol
x2: max_leaf_nodes
−0.2512
−0.2528
−0.2544
−0.2560
−0.2576
−0.2592
x2: max_leaf_nodes
x9: tol

Surrogate
x6: max_bins
x4: min_samples_leaf
240
220
200
180
160
140
2
4
6
8
10
−0.216
−0.228
−0.240
−0.252
−0.264
−0.276

−0.22
−0.24
−0.26
250
200
150
x6: max_bins
2
4
6
8
10
x4: min_samples_leaf

Surrogate
x7: early_stopping
x4: min_samples_leaf
1.0
0.8
0.6
0.4
0.2
0.0
2
4
6
8
10
−0.195
−0.210
−0.225
−0.240
−0.255
−0.270
−0.285
−0.300
−0.315

−0.225
−0.250
−0.275
−0.300
1.00
0.75
0.50
0.25
0.00
x7: early_stopping
2
4
6
8
10
x4: min_samples_leaf

Surrogate
x8: n_iter_no_change
x4: min_samples_leaf
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2
4
6
8
10
−0.216
−0.228
−0.240
−0.252
−0.264
−0.276

−0.22
−0.24
−0.26
20
15
10
5
x8: n_iter_no_change
2
4
6
8
10
x4: min_samples_leaf

Surrogate
x9: tol
x4: min_samples_leaf
0.0010
0.0008
0.0006
0.0004
0.0002
2
4
6
8
10
−0.216
−0.228
−0.240
−0.252
−0.264
−0.276

x4: min_samples_leaf
x9: tol
−0.22
−0.24
−0.26
0.00100
0.00075
0.00050
0.00025
0.00000

Surrogate
x7: early_stopping
x6: max_bins
1.0
0.8
0.6
0.4
0.2
0.0
150
175
200
225
250
−0.256
−0.258
−0.260
−0.262
−0.264
−0.266
−0.268
−0.270
−0.272
−0.274

x6: max_bins
x7: early_stopping
−0.260
−0.265
−0.270
150
200
250
1.00
0.75
0.50
0.25
0.00

Surrogate
x8: n_iter_no_change
x6: max_bins
20.0
17.5
15.0
12.5
10.0
7.5
5.0
150
175
200
225
250
−0.2552
−0.2560
−0.2568
−0.2576
−0.2584
−0.2592
−0.2600

x6: max_bins
x8: n_iter_no_change
−0.256
−0.257
−0.258
−0.259
150
200
250
20
15
10
5

Surrogate
x9: tol
x6: max_bins
0.0010
0.0008
0.0006
0.0004
0.0002
150
175
200
225
250
−0.2526
−0.2538
−0.2550
−0.2562
−0.2574
−0.2586
0.254
0.256
0.258
0.00100
0.00075
0.00050
0.00025
0.00000

Surrogate
x8: n_iter_no_change
x7: early_stopping
20.0
17.5
15.0
12.5
10.0
7.5
5.0
0.0
0.2
0.4
0.6
0.8
1.0
−0.256
−0.260
−0.264
−0.268
−0.272
−0.276
0.260
0.265
0.270
20
15
10
5
0.00
0.25
0.50
0.75
1.00

Surrogate
x9: tol
x7: early_stopping
0.0010
0.0008
0.0006
0.0004
0.0002
0.0
0.2
0.4
0.6
0.8
1.0
−0.252
−0.256
−0.260
−0.264
−0.268
−0.272
0.255
0.260
0.265
0.270
0.00100
0.00075
0.00050
0.00025
0.00000
0.00
0.25
0.50
0.75
1.00

21.10.10 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
```

21.10.11 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)
```

22 HPT: sklearn SVC VBDP Data

This document refers to the following software versions:

- python: 3.10.10

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

```
spotPython                  0.2.38
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
```

22.1 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_bartz09_1min_5init_2023-06-19_00-50-20
```

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

22.2 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")
```

22.3 Step 3: PyTorch Data Loading

22.3.1 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`.

22.3.2 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})
```

22.4 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(),
#     )
```

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

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

`fun_control = add_core_model_to_fun_control(SVC, fun_control, SklearnHyperDict)`

Other core_models are, e.g.,:

- RidgeCV
- GradientBoostingRegressor
- ElasticNet
- RandomForestClassifier
- LogisticRegression
- KNeighborsClassifier
- RandomForestClassifier
- GradientBoostingClassifier
- HistGradientBoostingClassifier

We will use the `RandomForestClassifier` classifier in this example.

```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
fun_control = add_core_model_to_fun_control(core_model=core_model,
                              fun_control=fun_control,
                              hyper_dict=SklearnHyperDict,
                              filename=None)
```

Now `fun_control` has the information from the JSON file. The available hyperparameters are:

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

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

`break_ties`

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

22.6.1 Modify hyperparameter of type numeric and integer (boolean)

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

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

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

22.6.2 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 19.6.

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

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

The new setting can be controlled via:

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

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

22.6.3 Optimizers

Optimizers are described in Section 19.6.1.

22.6.4 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.

22.7 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"`.

22.7.1 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.

22.7.1.1 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}`.

22.7.1.2 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},
               })
```

22.7.2 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",
})
```

22.7.2.1 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,
# })
```

22.8 Step 8: Calling the SPOT Function

22.8.1 Preparing the SPOT Call

- Get types and variable names as well as lower and upper bounds for the hyperparameters.

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

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

```
| name        | type   | default   |    lower |   upper | transform   |
|-------------|--------|-----------|----------|---------|-------------|
| C           | float  | 1.0       |   0.1    |   10    | None        |
| kernel      | factor | rbf       |   0      |    0    | None        |
| degree      | int    | 3         |   3      |    3    | None        |
| gamma       | factor | scale     |   0      |    1    | None        |
| coef0       | float  | 0.0       |   0      |    0    | None        |
| shrinking   | factor | 0         |   0      |    1    | None        |
| probability | factor | 0         |   1      |    1    | None        |
| tol         | float  | 0.001     |   0.0001 |    0.01 | None        |
| cache_size  | float  | 200.0     | 100      |  400    | None        |
| break_ties  | factor | 0         |   0      |    1    | None        |
```

22.8.2 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
```

22.8.3 Run the `Spot` Optimizer

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

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

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

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

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

```
spotPython tuning: -0.37844611528822053 [----------] 0.29%

spotPython tuning: -0.37844611528822053 [----------] 0.59%

spotPython tuning: -0.37844611528822053 [----------] 0.98%

spotPython tuning: -0.37844611528822053 [----------] 1.36%

spotPython tuning: -0.37844611528822053 [----------] 1.72%

spotPython tuning: -0.37844611528822053 [----------] 2.15%

spotPython tuning: -0.37844611528822053 [----------] 2.94%

spotPython tuning: -0.37844611528822053 [----------] 3.47%

spotPython tuning: -0.3822055137844611 [----------] 4.19%

spotPython tuning: -0.3822055137844611 [----------] 4.90%

spotPython tuning: -0.3822055137844611 [#---------] 5.35%

spotPython tuning: -0.3822055137844611 [#---------] 5.97%
```

```
spotPython tuning: -0.3822055137844611 [#---------] 7.49%

spotPython tuning: -0.3822055137844611 [#---------] 9.32%

spotPython tuning: -0.3822055137844611 [#---------] 10.52%

spotPython tuning: -0.3822055137844611 [#---------] 11.60%

spotPython tuning: -0.3822055137844611 [#---------] 12.75%

spotPython tuning: -0.3822055137844611 [#---------] 13.73%

spotPython tuning: -0.3822055137844611 [#---------] 14.75%

spotPython tuning: -0.3897243107769423 [##--------] 16.03%

spotPython tuning: -0.3897243107769423 [##--------] 17.02%

spotPython tuning: -0.3897243107769423 [##--------] 18.45%

spotPython tuning: -0.3897243107769423 [##--------] 19.62%

spotPython tuning: -0.3897243107769423 [##--------] 20.51%

spotPython tuning: -0.3897243107769423 [##--------] 21.51%

spotPython tuning: -0.3897243107769423 [##--------] 22.59%

spotPython tuning: -0.3897243107769423 [##--------] 23.72%

spotPython tuning: -0.3897243107769423 [###-------] 25.24%

spotPython tuning: -0.3897243107769423 [###-------] 26.72%

spotPython tuning: -0.3897243107769423 [###-------] 27.53%

spotPython tuning: -0.3897243107769423 [###-------] 28.63%
```

```
spotPython tuning: -0.39598997493734334 [###-------] 29.62%

spotPython tuning: -0.39598997493734334 [###-------] 31.34%

spotPython tuning: -0.39598997493734334 [###-------] 33.05%

spotPython tuning: -0.39598997493734334 [###-------] 34.81%

spotPython tuning: -0.39598997493734334 [####------] 36.83%

spotPython tuning: -0.39598997493734334 [####------] 39.10%

spotPython tuning: -0.39598997493734334 [####------] 40.95%

spotPython tuning: -0.39598997493734334 [####------] 42.58%

spotPython tuning: -0.39598997493734334 [####------] 44.40%

spotPython tuning: -0.39598997493734334 [#####-----] 46.36%

spotPython tuning: -0.39598997493734334 [#####-----] 48.18%

spotPython tuning: -0.39598997493734334 [#####-----] 49.83%

spotPython tuning: -0.39598997493734334 [#####-----] 52.56%

spotPython tuning: -0.39598997493734334 [#####-----] 54.52%

spotPython tuning: -0.39598997493734334 [######----] 55.74%

spotPython tuning: -0.39598997493734334 [######----] 57.36%

spotPython tuning: -0.39598997493734334 [######----] 58.91%

spotPython tuning: -0.39598997493734334 [######----] 60.49%

spotPython tuning: -0.39598997493734334 [######----] 62.11%
```

```
spotPython tuning: -0.39598997493734334 [######----] 63.44%

spotPython tuning: -0.39598997493734334 [#######---] 65.01%

spotPython tuning: -0.39598997493734334 [#######---] 66.71%

spotPython tuning: -0.39598997493734334 [#######---] 68.42%

spotPython tuning: -0.39598997493734334 [#######---] 70.03%

spotPython tuning: -0.39598997493734334 [#######---] 71.56%

spotPython tuning: -0.39598997493734334 [#######---] 73.03%

spotPython tuning: -0.39598997493734334 [#######---] 74.53%

spotPython tuning: -0.39598997493734334 [########--] 77.34%

spotPython tuning: -0.39598997493734334 [########--] 80.83%

spotPython tuning: -0.39598997493734334 [########--] 84.78%

spotPython tuning: -0.39598997493734334 [#########-] 88.66%

spotPython tuning: -0.39598997493734334 [#########-] 92.50%

spotPython tuning: -0.39598997493734334 [##########] 96.21%

spotPython tuning: -0.39598997493734334 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x284a73730>
```

22.9 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section 19.9, see also the description in the documentation: Tensorboard.

22.10 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 22.1: 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 |   1.0858799370314682 | 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 |                   0.0 | None
| coef0       | float  | 0.0       |     0.0 |     0.0 |                   0.0 | None
| shrinking   | factor | 0         |     0.0 |     1.0 |                   0.0 | None
| probability | factor | 0         |     1.0 |     1.0 |                   1.0 | None
| tol         | float  | 0.001     |  0.0001 |    0.01 | 0.007236213886936429 | None
| cache_size  | float  | 200.0     |   100.0 |   400.0 |   223.25023402218324 | None
| break_ties  | factor | 0         |     0.0 |     1.0 |                   0.0 | None
```

22.10.1 Show variable importance

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

Figure 22.2: Variable importance plot, threshold 0.025.

22.10.2 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})
> ```

22.10.3 Get SPOT Results

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

```
[[1.08587994e+00 0.00000000e+00 3.00000000e+00 0.00000000e+00
  0.00000000e+00 0.00000000e+00 1.00000000e+00 7.23621389e-03
  2.23250234e+02 0.00000000e+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': 1.0858799370314682,
  'kernel': 'rbf',
  'degree': 3,
  'gamma': 'scale',
  'coef0': 0.0,
```

```
  'shrinking': 0,
  'probability': 1,
  'tol': 0.007236213886936429,
  'cache_size': 223.25023402218324,
  'break_ties': 0}]
```

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

```
SVC(C=1.0858799370314682, break_ties=0, cache_size=223.25023402218324,
    probability=1, shrinking=0, tol=0.007236213886936429)
```

22.10.4 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
```

```
0.3870056497175141
```

```
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
```

22.10.5 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: 0.3849968612680478
std_res: 0.002894448683841439
min_res: 0.37947269303201503
max_res: 0.3907721280602636
median_res: 0.38465160075329563
```

22.10.6 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)
```

```
0.3898305084745763
```

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: 0.3841807909604519
std_res: 0.0048503311417656915
min_res: 0.3747645951035781
max_res: 0.39453860640301314
median_res: 0.384180790960452
```

22.10.7 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)
```

```
(-0.39598997493734334, -0.3446115288220551)
```

22.10.8 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)
```

```
(0.36886792452830186, None)
```

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

```
(0.35778867102396517, 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)
```

```
(0.36836016096579477, None)
```

22.10.9 Detailed Hyperparameter Plots

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

```
C:  2.944372292822479
gamma:  100.0
tol:  0.3630375999728226
cache_size:  0.39465556083146097
```

Surrogate
x1: gamma
x0: C
−0.357
−0.360
−0.363
−0.366
−0.369
−0.372
x0: C
x1: gamma

Surrogate
x3: tol
x0: C
−0.356
−0.360
−0.364
−0.368
−0.372
−0.376
x0: C
x3: tol

Surrogate
x4: cache_size
x0: C
−0.3555
−0.3585
−0.3615
−0.3645
−0.3675
−0.3705
x0: C
x4: cache_size

Surrogate
x3: tol
x1: gamma
0.010
0.008
0.006
0.004
0.002
0.0
0.2
0.4
0.6
0.8
1.0
−0.3576
−0.3592
−0.3608
−0.3624
−0.3640
−0.3656
0.360
0.362
0.364
0.0100
0.0075
0.0050
0.0025
0.0000
x3: tol
0.00
0.25
0.50
0.75
1.00
x1: gamma

Surrogate
x4: cache_size
x1: gamma
400
350
300
250
200
150
100
0.0
0.2
0.4
0.6
0.8
1.0
−0.3576
−0.3588
−0.3600
−0.3612
−0.3624
−0.3636
−0.3648
0.358
0.360
0.362
0.364
400
300
200
100
x4: cache_size
0.00
0.25
0.50
0.75
1.00
x1: gamma

Surrogate
x4: cache_size
x3: tol
400
350
300
250
200
150
100
0.002
0.004
0.006
0.008
0.010
−0.3576
−0.3584
−0.3592
−0.3600
−0.3608
−0.3616
0.358
0.359
0.360
0.361
400
300
200
100
x4: cache_size
0.0000
0.0025
0.0050
0.0075
0.0100
x3: tol

22.10.10 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
```

22.10.11 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)
```

23 HPT: sklearn KNN Classifier VBDP Data

This document refers to the following software versions:

- python: 3.10.10

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

```
spotPython                  0.2.38
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
```

23.1 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_bartz09_1min_5init_2023-06-19_00-52-52
```

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

23.2 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")
```

23.2.1 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`.

23.2.2 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})
```

23.3 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(),
#     )
```

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

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

```
fun_control = add_core_model_to_fun_control(SVC, fun_control, SklearnHyperDict)
```

Other core_models are, e.g.,:

- RidgeCV
- GradientBoostingRegressor
- ElasticNet
- RandomForestClassifier
- LogisticRegression
- KNeighborsClassifier
- RandomForestClassifier
- GradientBoostingClassifier
- HistGradientBoostingClassifier

We will use the `RandomForestClassifier` classifier in this example.

```
from sklearn.linear_model import RidgeCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.linear_model import ElasticNet
```

```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
fun_control = add_core_model_to_fun_control(core_model=core_model,
                              fun_control=fun_control,
                              hyper_dict=SklearnHyperDict,
                              filename=None)
```

Now `fun_control` has the information from the JSON file. The available hyperparameters are:

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

```
n_neighbors
weights
algorithm
leaf_size
p
```

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

23.5.1 Modify hyperparameter of type numeric and integer (boolean)

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

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

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

23.5.2 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 19.6.

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

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

The new setting can be controlled via:

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

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

23.5.3 Optimizers

Optimizers are described in Section 19.6.1.

23.5.4 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.

23.6 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"`.

23.6.1 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.

23.6.1.1 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}`.

23.6.1.2 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},
               })
```

23.6.2 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",
})
```

23.6.2.1 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,
# })
```

23.7 Step 8: Calling the SPOT Function

23.7.1 Preparing the SPOT Call

- Get types and variable names as well as lower and upper bounds for the hyperparameters.

```
# extract the variable types, names, and bounds
from spotPython.hyperparameters.values import (get_bound_values,
    get_var_name,
```

```
    get_var_type,)
var_type = get_var_type(fun_control)
var_name = get_var_name(fun_control)
fun_control.update({"var_type": var_type,
                    "var_name": var_name})
lower = get_bound_values(fun_control, "lower")
upper = get_bound_values(fun_control, "upper")
```

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

```
| name        | type   | default   |   lower |   upper | transform             |
|-------------|--------|-----------|---------|---------|-----------------------|
| n_neighbors | int    | 2         |       1 |       7 | transform_power_2_int |
| weights     | factor | uniform   |       0 |       1 | None                  |
| algorithm   | factor | auto      |       0 |       3 | None                  |
| leaf_size   | int    | 5         |       2 |       7 | transform_power_2_int |
| p           | int    | 2         |       1 |       2 | None                  |
```

23.7.2 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
```

23.7.3 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
hyper_dict=SklearnHyperDict().load()
X_start = get_default_hyperparameters_as_array(fun_control, hyper_dict)
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 [----------] 0.51%

spotPython tuning: -0.3107769423558897 [----------] 0.77%

spotPython tuning: -0.3107769423558897 [----------] 1.02%

spotPython tuning: -0.3107769423558897 [----------] 1.27%
```

```
spotPython tuning: -0.3107769423558897 [----------] 1.58%

spotPython tuning: -0.3107769423558897 [----------] 1.95%

spotPython tuning: -0.3107769423558897 [----------] 2.28%

spotPython tuning: -0.3107769423558897 [----------] 2.60%

spotPython tuning: -0.3107769423558897 [----------] 2.89%

spotPython tuning: -0.3107769423558897 [----------] 3.18%

spotPython tuning: -0.3107769423558897 [----------] 4.16%

spotPython tuning: -0.3107769423558897 [#---------] 5.15%

spotPython tuning: -0.3107769423558897 [#---------] 6.32%

spotPython tuning: -0.3107769423558897 [#---------] 7.48%

spotPython tuning: -0.3107769423558897 [#---------] 8.73%

spotPython tuning: -0.3107769423558897 [#---------] 10.21%

spotPython tuning: -0.3107769423558897 [#---------] 11.39%

spotPython tuning: -0.3107769423558897 [#---------] 13.42%

spotPython tuning: -0.3107769423558897 [#---------] 14.75%

spotPython tuning: -0.3107769423558897 [##--------] 15.85%

spotPython tuning: -0.3107769423558897 [##--------] 16.75%

spotPython tuning: -0.3107769423558897 [##--------] 17.77%

spotPython tuning: -0.3107769423558897 [##--------] 19.01%
```

```
spotPython tuning: -0.3107769423558897 [##--------] 19.95%

spotPython tuning: -0.3107769423558897 [##--------] 21.17%

spotPython tuning: -0.3107769423558897 [##--------] 22.48%

spotPython tuning: -0.3107769423558897 [##--------] 23.99%

spotPython tuning: -0.3107769423558897 [###-------] 25.65%

spotPython tuning: -0.3107769423558897 [###-------] 27.42%

spotPython tuning: -0.3107769423558897 [###-------] 29.04%

spotPython tuning: -0.3107769423558897 [###-------] 31.40%

spotPython tuning: -0.3107769423558897 [###-------] 33.46%

spotPython tuning: -0.3107769423558897 [####------] 35.99%

spotPython tuning: -0.3107769423558897 [####------] 39.89%

spotPython tuning: -0.3107769423558897 [####------] 42.94%

spotPython tuning: -0.3107769423558897 [#####-----] 45.82%

spotPython tuning: -0.3107769423558897 [#####-----] 48.87%

spotPython tuning: -0.3107769423558897 [#####-----] 51.58%

spotPython tuning: -0.3107769423558897 [#####-----] 54.29%

spotPython tuning: -0.3107769423558897 [######----] 57.27%

spotPython tuning: -0.3107769423558897 [######----] 60.30%

spotPython tuning: -0.3107769423558897 [######----] 63.62%
```

```
spotPython tuning: -0.3107769423558897 [#######---] 66.83%

spotPython tuning: -0.3107769423558897 [#######---] 69.73%

spotPython tuning: -0.3107769423558897 [#######---] 72.95%

spotPython tuning: -0.3107769423558897 [########--] 77.43%

spotPython tuning: -0.3107769423558897 [########--] 80.66%

spotPython tuning: -0.3107769423558897 [########--] 83.92%

spotPython tuning: -0.3107769423558897 [#########-] 87.50%

spotPython tuning: -0.3107769423558897 [#########-] 91.41%

spotPython tuning: -0.3107769423558897 [#########-] 94.68%

spotPython tuning: -0.3107769423558897 [##########] 97.17%

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

<spotPython.spot.spot.Spot at 0x1376e9690>
```

23.8 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section 19.9, see also the description in the documentation: Tensorboard.

23.9 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 23.1: 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                  |
```

23.9.1 Show variable importance

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

Figure 23.2: Variable importance plot, threshold 0.025.

23.9.2 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))])
```

23.9.3 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')
```

23.9.4 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
```

23.9.5 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
```

23.9.6 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
```

23.9.7 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)
```

23.9.8 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)
```

23.9.9 Detailed Hyperparameter Plots

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

```
n_neighbors:  7.659298853276286
weights:  100.0
```

23.9.10 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
```

23.9.11 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)
```

24 HPT PyTorch: Regression

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

This document refers to the following software versions:

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

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

```
spotPython                  0.2.38
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
```

24.1 Step 1: Setup

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

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

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

Note: Device selection

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

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

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

```
cpu
```

```
import os
import copy
import socket
from datetime import datetime
from dateutil.tz import tzlocal
start_time = datetime.now(tzlocal())
HOSTNAME = socket.gethostname().split(".")[0]
experiment_name = '24-torch' + "_" + HOSTNAME + "_" + str(MAX_TIME) + "min_" + str(INIT_SI
experiment_name = experiment_name.replace(':', '-')
print(experiment_name)
if not os.path.exists('./figures'):
    os.makedirs('./figures')
```

```
24-torch_bartz09_1min_5init_2023-06-19_00-57-39
```

24.2 Step 2: Initialization of the `fun_control` Dictionary

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

`spotPython` uses a Python dictionary for storing the information required for the hyperparameter tuning process, which was described in Section 19.2.

```
from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="regression",
 tensorboard_path="runs/24_spot_torch_regression",
 device=DEVICE)
```

24.3 Step 3: PyTorch Data Loading

```
# Create dataset
import pandas as pd
import numpy as np
from sklearn import datasets as sklearn_datasets
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
X, y = sklearn_datasets.make_regression(
    n_samples=1000, n_features=10, noise=1, random_state=123)
y = y.reshape(-1, 1)

# Normalize the data
X_scaler = MinMaxScaler()
X_scaled = X_scaler.fit_transform(X)
y_scaler = MinMaxScaler()
y_scaled = y_scaler.fit_transform(y)

# combine the features and target into a single dataframe named train_df
train_df = pd.DataFrame(np.hstack((X_scaled, y_scaled)))

target_column = "y"
n_samples = train_df.shape[0]
n_features = train_df.shape[1] - 1
```

```
train_df.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
X_train, X_test, y_train, y_test = train_test_split(train_df.drop(target_column,
    axis=1),
    train_df[target_column],
    random_state=42,
    test_size=0.25)
trainset = pd.DataFrame(np.hstack((X_train, np.array(y_train).reshape(-1, 1))))
testset = pd.DataFrame(np.hstack((X_test, np.array(y_test).reshape(-1, 1))))
trainset.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
testset.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
print(train_df.shape)
print(trainset.shape)
print(testset.shape)
```

```
(1000, 11)
(750, 11)
(250, 11)
```

```
import torch
from spotPython.torch.dataframedataset import DataFrameDataset
dtype_x = torch.float32
dtype_y = torch.float32
train_df = DataFrameDataset(train_df, target_column=target_column,
    dtype_x=dtype_x, dtype_y=dtype_y)
train = DataFrameDataset(trainset, target_column=target_column,
    dtype_x=dtype_x, dtype_y=dtype_y)
test = DataFrameDataset(testset, target_column=target_column,
    dtype_x=dtype_x, dtype_y=dtype_y)
n_samples = len(train)
```

- Now we can test the data loading:

```
from spotPython.torch.traintest import create_train_val_data_loaders
trainloader, testloader = create_train_val_data_loaders(train, 2, True, 0)
for i, data in enumerate(trainloader, 0):
    inputs, labels = data
    print(inputs.shape)
    print(labels.shape)
    print(inputs)
    print(labels)
    break
```

```
torch.Size([2, 10])
torch.Size([2])
tensor([[0.6777, 0.5245, 0.5961, 0.5443, 0.4265, 0.4757, 0.6184, 0.4943, 0.8057,
         0.6815],
        [0.5407, 0.6625, 0.6278, 0.6597, 0.5130, 0.3093, 0.5147, 0.7693, 0.6450,
         0.3256]])
tensor([0.6539, 0.6178])
```

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

```
# add the dataset to the fun_control
fun_control.update({"data": train_df, # full dataset,
               "train": train,
               "test": test,
               "n_samples": n_samples,
               "target_column": target_column,})
```

24.4 Step 4: Specification of the Preprocessing Model

After the training and test data are specified and added to the `fun_control` dictionary, `spotPython` allows the specification of a data preprocessing pipeline, e.g., for the scaling of the data or for the one-hot encoding of categorical variables, see Section 19.4. This feature is not used here, so we do not change the default value (which is `None`).

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

24.5.1 Implementing a Configurable Neural Network With spotPython

`spotPython` includes the `Net_lin_reg` class which is implemented in the file `netregression.py`.

This class inherits from the class `Net_Core` which is implemented in the file `netcore.py`, see Section 19.5.1.

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


class Net_lin_reg(netcore.Net_Core):
    def __init__(
```

```
        self, _L_in, _L_out, l1, dropout_prob, lr_mult,
        batch_size, epochs, k_folds, patience, optimizer,
        sgd_momentum
    ):
        super(Net_lin_reg, self).__init__(
            lr_mult=lr_mult,
            batch_size=batch_size,
            epochs=epochs,
            k_folds=k_folds,
            patience=patience,
            optimizer=optimizer,
            sgd_momentum=sgd_momentum,
        )
        l2 = max(l1 // 2, 4)
        self.fc1 = nn.Linear(_L_in, l1)
        self.fc2 = nn.Linear(l1, l2)
        self.fc3 = nn.Linear(l2, _L_out)
        self.relu = nn.ReLU()
        self.softmax = nn.Softmax(dim=1)
        self.dropout1 = nn.Dropout(p=dropout_prob)
        self.dropout2 = nn.Dropout(p=dropout_prob / 2)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.dropout1(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.dropout2(x)
        x = self.fc3(x)
        return x
```

24.5.1.1 The `Net_Core` class

`Net_lin_reg` inherits from the class `Net_Core` which is implemented in the file `netcore.py`. This class was described in Section 19.5.1.

```
from spotPython.torch.netregression import Net_lin_reg
from spotPython.data.torch_hyper_dict import TorchHyperDict
from spotPython.hyperparameters.values import add_core_model_to_fun_control
fun_control = add_core_model_to_fun_control(core_model=Net_lin_reg,
```

```
                                    fun_control=fun_control,
                                    hyper_dict=TorchHyperDict,
                                    filename=None)
```

24.5.2 The Search Space

24.5.3 Configuring the Search Space With spotPython

24.5.3.1 The `hyper_dict` Hyperparameters for the Selected Algorithm

`spotPython` uses `JSON` files for the specification of the hyperparameters, which were described in Section 19.5.5.

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

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

```
{'_L_in': {'type': 'int',
  'default': 10,
  'transform': 'None',
  'lower': 10,
  'upper': 10},
 '_L_out': {'type': 'int',
  'default': 1,
  'transform': 'None',
  'lower': 1,
  'upper': 1},
 'l1': {'type': 'int',
  'default': 3,
  'transform': 'transform_power_2_int',
  'lower': 3,
  'upper': 8},
 'dropout_prob': {'type': 'float',
  'default': 0.01,
  'transform': 'None',
  'lower': 0.0,
  'upper': 0.9},
 'lr_mult': {'type': 'float',
  'default': 1.0,
  'transform': 'None',
  'lower': 0.1,
```

```
 'upper': 10.0},
'batch_size': {'type': 'int',
 'default': 4,
 'transform': 'transform_power_2_int',
 'lower': 1,
 'upper': 4},
'epochs': {'type': 'int',
 'default': 4,
 'transform': 'transform_power_2_int',
 'lower': 4,
 'upper': 9},
'k_folds': {'type': 'int',
 'default': 1,
 'transform': 'None',
 'lower': 1,
 'upper': 1},
'patience': {'type': 'int',
 'default': 2,
 'transform': 'transform_power_2_int',
 'lower': 1,
 'upper': 5},
'optimizer': {'levels': ['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}}
```

24.6 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 19.6.

24.6.1 Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model`

24.6.1.1 Modify Hyperparameters of Type numeric and integer (boolean)

```
from spotPython.hyperparameters.values import modify_hyper_parameter_bounds

fun_control = modify_hyper_parameter_bounds(fun_control, "epochs", bounds=[2, 16])
fun_control = modify_hyper_parameter_bounds(fun_control, "patience", bounds=[3, 7])
```

24.6.1.2 Modify Hyperparameter of Type factor

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


fun_control.update({
               "_L_in": n_features,
               "_L_out": 1,})
```

24.6.2 Optimizers

Optimizers are described in Section 19.6.1.

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

24.7.1 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 19.7.1)
2. the loss function (and a metric).

24.7.2 Loss Functions and Metrics

The key `"loss_function"` specifies the loss function which is used during the optimization, see Section 19.7.5.

We will use MSE loss for the regression task.

```
from torch.nn import MSELoss
loss_torch = MSELoss()
fun_control.update({"loss_function": loss_torch})
```

24.7.3 Metric

```
from torchmetrics import MeanAbsoluteError
metric_torch = MeanAbsoluteError().to(fun_control["device"])
fun_control.update({"metric_torch": metric_torch})
```

24.8 Step 8: Calling the SPOT Function

24.8.1 Preparing the SPOT Call

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

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

```
                    "var_name": var_name})
lower = get_bound_values(fun_control, "lower")
upper = get_bound_values(fun_control, "upper")
```

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

```
| name         | type   | default   |   lower |   upper | transform             |
|--------------|--------|-----------|---------|---------|-----------------------|
| _L_in        | int    | 10        |    10   |    10   | None                  |
| _L_out       | int    | 1         |     1   |     1   | None                  |
| l1           | int    | 3         |     3   |     8   | transform_power_2_int |
| dropout_prob | float  | 0.01      |     0   |     0.9 | None                  |
| lr_mult      | float  | 1.0       |     0.1 |    10   | None                  |
| batch_size   | int    | 4         |     1   |     4   | transform_power_2_int |
| epochs       | int    | 4         |     2   |    16   | transform_power_2_int |
| k_folds      | int    | 1         |     1   |     1   | None                  |
| patience     | int    | 2         |     3   |     7   | transform_power_2_int |
| optimizer    | factor | SGD       |     0   |     6   | None                  |
| sgd_momentum | float  | 0.0       |     0   |     1   | None                  |
```

24.8.2 The Objective Function `fun_torch`

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

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

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

24.8.3 Starting the Hyperparameter Tuning

The `spotPython` hyperparameter tuning is started by calling the `Spot` function as described in Section 19.8.4.

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

```
config: {'_L_in': 10, '_L_out': 1, 'l1': 64, 'dropout_prob': 0.7103122166156, 'lr_mult': 3.62
Epoch: 1 |

MeanAbsoluteError: 0.1466274559497833 | Loss: 0.0353190445855848 | Epoch: 2 | MeanAbsoluteErr

MeanAbsoluteError: 0.1220633313059807 | Loss: 0.0233279193604463 | Epoch: 10 |

MeanAbsoluteError: 0.1305875182151794 | Loss: 0.0255719802072762 | Epoch: 11 |
```

```
MeanAbsoluteError: 0.1181919872760773 | Loss: 0.0217432543929470 | Epoch: 12 | MeanAbsoluteEr

MeanAbsoluteError: 0.1135564446449280 | Loss: 0.0194539563656834 | Epoch: 21 |

MeanAbsoluteError: 0.1079207509756088 | Loss: 0.0179831690865716 | Epoch: 22 |

MeanAbsoluteError: 0.0965467616915703 | Loss: 0.0156055252239304 | Epoch: 23 | MeanAbsoluteEr

MeanAbsoluteError: 0.0870922952890396 | Loss: 0.0117197299412893 |

Epoch: 32 | MeanAbsoluteError: 0.0850548893213272 | Loss: 0.0119972021784633 | Epoch: 33 |

MeanAbsoluteError: 0.0778444707393646 | Loss: 0.0102436203681129 | Epoch: 34 | MeanAbsoluteEr

MeanAbsoluteError: 0.0908489823341370 | Loss: 0.0140849871340355 | Epoch: 43 | MeanAbsoluteEr

MeanAbsoluteError: 0.0794065743684769 | Loss: 0.0103051059046074 | Epoch: 45 | MeanAbsoluteEr

MeanAbsoluteError: 0.0764234811067581 | Loss: 0.0098889790058724 | Epoch: 54 | MeanAbsoluteEr

MeanAbsoluteError: 0.0714949071407318 | Loss: 0.0086169248373297 | Epoch: 56 | MeanAbsoluteEr

MeanAbsoluteError: 0.0793527886271477 | Loss: 0.0100108544697593 | Epoch: 65 | MeanAbsoluteEr

MeanAbsoluteError: 0.0697535350918770 | Loss: 0.0086653030379430 | Epoch: 67 | MeanAbsoluteEr
Returned to Spot: Validation loss: 0.008376560406759381

config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.19981931523998656, 'lr_mult':
Epoch: 1 | MeanAbsoluteError: 0.1673418134450912 | Loss: 0.0424544391663451 | Epoch: 2 | Mean

MeanAbsoluteError: 0.1122188344597816 | Loss: 0.0177003619212069 | Epoch: 13 | MeanAbsoluteEr

MeanAbsoluteError: 0.0795300900936127 | Loss: 0.0102928119564527 | Epoch: 16 | MeanAbsoluteEr

MeanAbsoluteError: 0.1188381612300873 | Loss: 0.0168768356701261 | Epoch: 32 | MeanAbsoluteEr

MeanAbsoluteError: 0.0490641072392464 | Loss: 0.0044612448025299 | Epoch: 35 | MeanAbsoluteEr
```

```
MeanAbsoluteError: 0.1186487600207329 | Loss: 0.0172833696890034 | Epoch: 51 | MeanAbsoluteEr

MeanAbsoluteError: 0.1021685376763344 | Loss: 0.0161694923513814 | Epoch: 54 | MeanAbsoluteEr

MeanAbsoluteError: 0.0408714860677719 | Loss: 0.0026034391117527 | Epoch: 70 | MeanAbsoluteEr

MeanAbsoluteError: 0.0404348894953728 | Loss: 0.0031548079630164 | Epoch: 73 | MeanAbsoluteEr

MeanAbsoluteError: 0.0771021768450737 | Loss: 0.0095599651287653 | Epoch: 89 | MeanAbsoluteEr

MeanAbsoluteError: 0.0399103760719299 | Loss: 0.0027846394223161 | Epoch: 92 | MeanAbsoluteEr

MeanAbsoluteError: 0.0744219794869423 | Loss: 0.0070036347024143 | Epoch: 108 | MeanAbsolute

MeanAbsoluteError: 0.0533020980656147 | Loss: 0.0047700274490604 | Epoch: 111 | MeanAbsolute

MeanAbsoluteError: 0.0850306451320648 | Loss: 0.0090708475510933 | Epoch: 127 | MeanAbsolute

MeanAbsoluteError: 0.0783797204494476 | Loss: 0.0079414817553602 | Epoch: 130 | MeanAbsolute

MeanAbsoluteError: 0.0652107521891594 | Loss: 0.0052502226010945 | Epoch: 146 | MeanAbsolute
Returned to Spot: Validation loss: 0.006078019811722793

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

MeanAbsoluteError: 0.1906510591506958 | Loss: 0.0566130275760467 | Epoch: 2 | MeanAbsoluteEr

MeanAbsoluteError: 0.1716860234737396 | Loss: 0.0453533054911531 | Epoch: 4 |

MeanAbsoluteError: 0.1642833799123764 | Loss: 0.0442047290702855 | Epoch: 5 | MeanAbsoluteEr

MeanAbsoluteError: 0.1520566493272781 | Loss: 0.0381441435176142 | Epoch: 7 |

MeanAbsoluteError: 0.1399800032377243 | Loss: 0.0327586910549629 | Epoch: 8 | MeanAbsoluteEr

MeanAbsoluteError: 0.1394905894994736 | Loss: 0.0313777056470280 | Epoch: 10 |
```

```
MeanAbsoluteError: 0.1395369917154312 | Loss: 0.0318273327293961 | Epoch: 11 | MeanAbsoluteEr

MeanAbsoluteError: 0.1421286910772324 | Loss: 0.0323932446304631 | Epoch: 13 |

MeanAbsoluteError: 0.1423172950744629 | Loss: 0.0326136383055321 | Epoch: 14 | MeanAbsoluteEr

MeanAbsoluteError: 0.1391330063343048 | Loss: 0.0315160674493382 | Epoch: 16 |

MeanAbsoluteError: 0.1370610892772675 | Loss: 0.0321110289019998 | Epoch: 17 | MeanAbsoluteEr

MeanAbsoluteError: 0.1361970901489258 | Loss: 0.0300712558169228 | Epoch: 19 |

MeanAbsoluteError: 0.1363485753536224 | Loss: 0.0301083630094460 | Epoch: 20 | MeanAbsoluteEr

MeanAbsoluteError: 0.1375680267810822 | Loss: 0.0305653620764497 | Epoch: 22 |

MeanAbsoluteError: 0.1347493082284927 | Loss: 0.0300543742692874 | Epoch: 23 | MeanAbsoluteEr

MeanAbsoluteError: 0.1357720345258713 | Loss: 0.0304554024550210 | Epoch: 25 |

MeanAbsoluteError: 0.1357821375131607 | Loss: 0.0302951228236392 | Epoch: 26 | MeanAbsoluteEr

MeanAbsoluteError: 0.1364869773387909 | Loss: 0.0297238391905557 | Epoch: 28 |

MeanAbsoluteError: 0.1349464803934097 | Loss: 0.0293439257603798 | Epoch: 29 | MeanAbsoluteEr

MeanAbsoluteError: 0.1330074220895767 | Loss: 0.0292786192549829 | Epoch: 31 |

MeanAbsoluteError: 0.1351703405380249 | Loss: 0.0287952701947749 | Epoch: 32 | MeanAbsoluteEr

MeanAbsoluteError: 0.1314263492822647 | Loss: 0.0291882812933181 | Epoch: 34 |

MeanAbsoluteError: 0.1285103857517242 | Loss: 0.0281552277073691 | Epoch: 35 | MeanAbsoluteEr

MeanAbsoluteError: 0.1359022408723831 | Loss: 0.0299255779484520 | Epoch: 37 |

MeanAbsoluteError: 0.1304128170013428 | Loss: 0.0280273155771041 | Epoch: 38 | MeanAbsoluteEr
```

```
MeanAbsoluteError: 0.1382595598697662 | Loss: 0.0306810739170639 | Epoch: 40 |

MeanAbsoluteError: 0.1321687102317810 | Loss: 0.0290237142948899 | Epoch: 41 | MeanAbsoluteEr

MeanAbsoluteError: 0.1369796991348267 | Loss: 0.0307540058142816 | Epoch: 43 |

MeanAbsoluteError: 0.1309502422809601 | Loss: 0.0283378096479767 | Epoch: 44 | MeanAbsoluteEr

MeanAbsoluteError: 0.1342477351427078 | Loss: 0.0293963913764067 | Epoch: 46 |

MeanAbsoluteError: 0.1317810565233231 | Loss: 0.0283242599003036 | Epoch: 47 | MeanAbsoluteEr

MeanAbsoluteError: 0.1321448385715485 | Loss: 0.0296947820054872 | Epoch: 49 |

MeanAbsoluteError: 0.1294513940811157 | Loss: 0.0284798011068293 | Epoch: 50 | MeanAbsoluteEr

MeanAbsoluteError: 0.1346157044172287 | Loss: 0.0287185958347012 | Epoch: 52 |

MeanAbsoluteError: 0.1309388875961304 | Loss: 0.0278205125815362 | Epoch: 53 | MeanAbsoluteEr

MeanAbsoluteError: 0.1319216340780258 | Loss: 0.0286977700029335 | Epoch: 55 |

MeanAbsoluteError: 0.1374904364347458 | Loss: 0.0299436161385772 | Epoch: 56 | MeanAbsoluteEr

MeanAbsoluteError: 0.1346812844276428 | Loss: 0.0294148891981846 | Epoch: 58 |

MeanAbsoluteError: 0.1346682608127594 | Loss: 0.0298371701462505 | Epoch: 59 | MeanAbsoluteEr

MeanAbsoluteError: 0.1298616826534271 | Loss: 0.0276389843689200 | Epoch: 61 |

MeanAbsoluteError: 0.1340327411890030 | Loss: 0.0297161840098003 | Epoch: 62 | MeanAbsoluteEr

MeanAbsoluteError: 0.1327998042106628 | Loss: 0.0297021915389147 | Epoch: 64 |

MeanAbsoluteError: 0.1320655047893524 | Loss: 0.0289745827367005 | Epoch: 65 | MeanAbsoluteEr

MeanAbsoluteError: 0.1323017925024033 | Loss: 0.0285230930376565 | Epoch: 67 |
```

```
MeanAbsoluteError: 0.1330378651618958 | Loss: 0.0293966636214949 | Epoch: 68 | MeanAbsoluteEr

MeanAbsoluteError: 0.1285994499921799 | Loss: 0.0273092211233840 | Epoch: 70 |

MeanAbsoluteError: 0.1319550871849060 | Loss: 0.0284430118997989 | Epoch: 71 | MeanAbsoluteEr

MeanAbsoluteError: 0.1346422731876373 | Loss: 0.0297366992243891 | Epoch: 73 |

MeanAbsoluteError: 0.1301766782999039 | Loss: 0.0272051566302495 | Epoch: 74 | MeanAbsoluteEr

MeanAbsoluteError: 0.1317600756883621 | Loss: 0.0291800176037941 | Epoch: 76 |

MeanAbsoluteError: 0.1346939802169800 | Loss: 0.0291340009025104 | Epoch: 77 | MeanAbsoluteEr

MeanAbsoluteError: 0.1309217363595963 | Loss: 0.0285290207299113 | Epoch: 79 |

MeanAbsoluteError: 0.1310009360313416 | Loss: 0.0287417869519171 | Epoch: 80 | MeanAbsoluteEr

MeanAbsoluteError: 0.1329780519008636 | Loss: 0.0285600786907404 | Epoch: 82 |

MeanAbsoluteError: 0.1344222277402878 | Loss: 0.0295988584190976 | Epoch: 83 | MeanAbsoluteEr

MeanAbsoluteError: 0.1298445463180542 | Loss: 0.0279037268961838 | Epoch: 85 |

MeanAbsoluteError: 0.1343757510185242 | Loss: 0.0294022890322473 | Epoch: 86 | MeanAbsoluteEr

MeanAbsoluteError: 0.1334653198719025 | Loss: 0.0296604452480096 | Epoch: 88 |

MeanAbsoluteError: 0.1317977160215378 | Loss: 0.0282055246009259 | Epoch: 89 | MeanAbsoluteEr

MeanAbsoluteError: 0.1291132122278214 | Loss: 0.0275769346447972 | Epoch: 91 |

MeanAbsoluteError: 0.1286529302597046 | Loss: 0.0272183367642841 | Epoch: 92 | MeanAbsoluteEr

MeanAbsoluteError: 0.1346822530031204 | Loss: 0.0295743861073667 | Epoch: 94 |

MeanAbsoluteError: 0.1293519586324692 | Loss: 0.0278089396092885 | Epoch: 95 | MeanAbsoluteEr
```

```
MeanAbsoluteError: 0.1330264508724213 | Loss: 0.0295699854003518 | Epoch: 97 |

MeanAbsoluteError: 0.1318452507257462 | Loss: 0.0281946755117193 | Epoch: 98 | MeanAbsoluteE

MeanAbsoluteError: 0.1315527558326721 | Loss: 0.0282325708988355 | Epoch: 100 |

MeanAbsoluteError: 0.1309109330177307 | Loss: 0.0279813521570516 | Epoch: 101 | MeanAbsolute

MeanAbsoluteError: 0.1337037980556488 | Loss: 0.0289984912661991 | Epoch: 103 |

MeanAbsoluteError: 0.1246166229248047 | Loss: 0.0271460152479510 | Epoch: 104 | MeanAbsolute

MeanAbsoluteError: 0.1319017112255096 | Loss: 0.0285892524283069 | Epoch: 106 |

MeanAbsoluteError: 0.1313085258007050 | Loss: 0.0286447932736094 | Epoch: 107 | MeanAbsolute

MeanAbsoluteError: 0.1300026625394821 | Loss: 0.0281772461375901 | Epoch: 109 |

MeanAbsoluteError: 0.1292364895343781 | Loss: 0.0273264225067881 | Epoch: 110 | MeanAbsolute

MeanAbsoluteError: 0.1338188201189041 | Loss: 0.0291431595859467 | Epoch: 112 |

MeanAbsoluteError: 0.1297620385885239 | Loss: 0.0277013270370662 | Epoch: 113 | MeanAbsolute

MeanAbsoluteError: 0.1278954893350601 | Loss: 0.0266686392491829 | Epoch: 115 |

MeanAbsoluteError: 0.1285887360572815 | Loss: 0.0276326698025029 | Epoch: 116 | MeanAbsolute

MeanAbsoluteError: 0.1331813931465149 | Loss: 0.0291553693435465 | Epoch: 118 |

MeanAbsoluteError: 0.1279483288526535 | Loss: 0.0273537791483977 | Epoch: 119 | MeanAbsolute

MeanAbsoluteError: 0.1267756670713425 | Loss: 0.0261667006054389 | Epoch: 121 |

MeanAbsoluteError: 0.1303852349519730 | Loss: 0.0272127553664905 | Epoch: 122 | MeanAbsolute

MeanAbsoluteError: 0.1279974728822708 | Loss: 0.0273285877949093 | Epoch: 124 |
```

```
MeanAbsoluteError: 0.1312306672334671 | Loss: 0.0285562622719832 | Epoch: 125 | MeanAbsoluteE

MeanAbsoluteError: 0.1298609226942062 | Loss: 0.0271880495060517 | Epoch: 127 |

MeanAbsoluteError: 0.1304007023572922 | Loss: 0.0277032456419935 | Epoch: 128 | MeanAbsoluteE

MeanAbsoluteError: 0.1309804022312164 | Loss: 0.0274701611030226 | Epoch: 130 |

MeanAbsoluteError: 0.1333596706390381 | Loss: 0.0296031904340877 | Epoch: 131 | MeanAbsoluteE

MeanAbsoluteError: 0.1290948241949081 | Loss: 0.0267291824473068 | Epoch: 133 |

MeanAbsoluteError: 0.1235921084880829 | Loss: 0.0261699620826433 | Epoch: 134 | MeanAbsoluteE

MeanAbsoluteError: 0.1261629164218903 | Loss: 0.0252107639272193 | Epoch: 136 |

MeanAbsoluteError: 0.1280498504638672 | Loss: 0.0274560375397171 | Epoch: 137 | MeanAbsoluteE

MeanAbsoluteError: 0.1326957345008850 | Loss: 0.0284395028838965 | Epoch: 139 |

MeanAbsoluteError: 0.1263322383165359 | Loss: 0.0265780668291457 | Epoch: 140 | MeanAbsoluteE

MeanAbsoluteError: 0.1293486207723618 | Loss: 0.0271486859624080 | Epoch: 142 |

MeanAbsoluteError: 0.1301159858703613 | Loss: 0.0271208324583131 | Epoch: 143 | MeanAbsoluteE

MeanAbsoluteError: 0.1305940002202988 | Loss: 0.0267767125629568 | Epoch: 145 |

MeanAbsoluteError: 0.1275585591793060 | Loss: 0.0264739708795484 | Epoch: 146 | MeanAbsoluteE

MeanAbsoluteError: 0.1293350160121918 | Loss: 0.0264357659349722 | Epoch: 148 |

MeanAbsoluteError: 0.1287278085947037 | Loss: 0.0266599251719284 | Epoch: 149 | MeanAbsoluteE

MeanAbsoluteError: 0.1304862648248672 | Loss: 0.0277842126715404 | Epoch: 151 |

MeanAbsoluteError: 0.1297014504671097 | Loss: 0.0285358558750886 | Epoch: 152 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.1320214271545410 | Loss: 0.0288105371511968 | Epoch: 154 |

MeanAbsoluteError: 0.1297477483749390 | Loss: 0.0276919884920062 | Epoch: 155 | MeanAbsolutel

MeanAbsoluteError: 0.1340585649013519 | Loss: 0.0293564373658349 | Epoch: 157 |

MeanAbsoluteError: 0.1291810274124146 | Loss: 0.0279277173132868 | Epoch: 158 | MeanAbsolutel

MeanAbsoluteError: 0.1293893456459045 | Loss: 0.0273832349703298 | Epoch: 160 |

MeanAbsoluteError: 0.1258026808500290 | Loss: 0.0260411290766206 | Epoch: 161 | MeanAbsolutel

MeanAbsoluteError: 0.1227259412407875 | Loss: 0.0247193491001478 | Epoch: 163 |

MeanAbsoluteError: 0.1322482377290726 | Loss: 0.0274905565675969 | Epoch: 164 | MeanAbsolutel

MeanAbsoluteError: 0.1256670802831650 | Loss: 0.0252152828584076 | Epoch: 166 |

MeanAbsoluteError: 0.1271947920322418 | Loss: 0.0267017466344017 | Epoch: 167 | MeanAbsolutel

MeanAbsoluteError: 0.1222105175256729 | Loss: 0.0251927848520791 | Epoch: 169 |

MeanAbsoluteError: 0.1247329041361809 | Loss: 0.0258795425099379 | Epoch: 170 | MeanAbsolutel

MeanAbsoluteError: 0.1256471574306488 | Loss: 0.0260598217044875 | Epoch: 172 |

MeanAbsoluteError: 0.1299345791339874 | Loss: 0.0273397332461900 | Epoch: 173 | MeanAbsolutel

MeanAbsoluteError: 0.1273670494556427 | Loss: 0.0270048201809792 | Epoch: 175 |

MeanAbsoluteError: 0.1280515938997269 | Loss: 0.0275075311497009 | Epoch: 176 | MeanAbsolutel

MeanAbsoluteError: 0.1317229121923447 | Loss: 0.0268912806523637 | Epoch: 178 |

MeanAbsoluteError: 0.1282253563404083 | Loss: 0.0263487456663764 | Epoch: 179 | MeanAbsolutel

MeanAbsoluteError: 0.1269359141588211 | Loss: 0.0267749317950802 | Epoch: 181 |
```

```
MeanAbsoluteError: 0.1216124221682549 | Loss: 0.0249034886583831 | Epoch: 182 | MeanAbsoluteE

MeanAbsoluteError: 0.1254811435937881 | Loss: 0.0258665555629220 | Epoch: 184 |

MeanAbsoluteError: 0.1292385011911392 | Loss: 0.0271006098319776 | Epoch: 185 | MeanAbsoluteE

MeanAbsoluteError: 0.1305601596832275 | Loss: 0.0263004001591374 | Epoch: 187 |

MeanAbsoluteError: 0.1227712482213974 | Loss: 0.0260232710780110 | Epoch: 188 | MeanAbsoluteE

MeanAbsoluteError: 0.1301685571670532 | Loss: 0.0269746949927442 | Epoch: 190 |

MeanAbsoluteError: 0.1299546808004379 | Loss: 0.0271196957222370 | Epoch: 191 | MeanAbsoluteE

MeanAbsoluteError: 0.1280435621738434 | Loss: 0.0266953587290626 | Epoch: 193 |

MeanAbsoluteError: 0.1291545778512955 | Loss: 0.0270281937714511 | Epoch: 194 | MeanAbsoluteE

MeanAbsoluteError: 0.1260731071233749 | Loss: 0.0265622766069040 | Epoch: 196 |

MeanAbsoluteError: 0.1306811869144440 | Loss: 0.0274924078037535 | Epoch: 197 | MeanAbsoluteE

MeanAbsoluteError: 0.1303652077913284 | Loss: 0.0271731297598065 | Epoch: 199 |

MeanAbsoluteError: 0.1229001209139824 | Loss: 0.0246745005645668 | Epoch: 200 | MeanAbsoluteE

MeanAbsoluteError: 0.1251192092895508 | Loss: 0.0261234354450486 | Epoch: 202 |

MeanAbsoluteError: 0.1263633072376251 | Loss: 0.0260844302755625 | Epoch: 203 | MeanAbsoluteE

MeanAbsoluteError: 0.1339864581823349 | Loss: 0.0281826138154914 | Epoch: 205 |

MeanAbsoluteError: 0.1274283975362778 | Loss: 0.0256653116860737 | Epoch: 206 | MeanAbsoluteE

MeanAbsoluteError: 0.1288847923278809 | Loss: 0.0274490477941678 | Epoch: 208 |

MeanAbsoluteError: 0.1220516934990883 | Loss: 0.0251615593982691 | Epoch: 209 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.1193395629525185 | Loss: 0.0238369472503488 | Epoch: 211 |

MeanAbsoluteError: 0.1258046776056290 | Loss: 0.0254130344909693 | Epoch: 212 | MeanAbsolutel

MeanAbsoluteError: 0.1250565946102142 | Loss: 0.0258027397975093 | Epoch: 214 |

MeanAbsoluteError: 0.1238245517015457 | Loss: 0.0250486532835445 | Epoch: 215 | MeanAbsolutel

MeanAbsoluteError: 0.1243383288383484 | Loss: 0.0249360205214665 | Epoch: 217 |

MeanAbsoluteError: 0.1236165463924408 | Loss: 0.0254272254896932 | Epoch: 218 | MeanAbsolutel

MeanAbsoluteError: 0.1237355172634125 | Loss: 0.0258154591498896 | Epoch: 220 |

MeanAbsoluteError: 0.1241203472018242 | Loss: 0.0243424645811319 | Epoch: 221 | MeanAbsolutel

MeanAbsoluteError: 0.1243355572223663 | Loss: 0.0252037720529188 | Epoch: 223 |

MeanAbsoluteError: 0.1295566558837891 | Loss: 0.0277265193796484 | Epoch: 224 | MeanAbsolutel

MeanAbsoluteError: 0.1228041201829910 | Loss: 0.0249811793106104 | Epoch: 226 |

MeanAbsoluteError: 0.1220282763242722 | Loss: 0.0236570830468554 | Epoch: 227 | MeanAbsolutel

MeanAbsoluteError: 0.1270113587379456 | Loss: 0.0260583428938601 | Epoch: 229 |

MeanAbsoluteError: 0.1180952340364456 | Loss: 0.0232157800834587 | Epoch: 230 | MeanAbsolutel

MeanAbsoluteError: 0.1247200369834900 | Loss: 0.0247487998032011 | Epoch: 232 |

MeanAbsoluteError: 0.1141832619905472 | Loss: 0.0221180114623470 | Epoch: 233 | MeanAbsolutel

MeanAbsoluteError: 0.1204415634274483 | Loss: 0.0242793727351818 | Epoch: 235 |

MeanAbsoluteError: 0.1235950440168381 | Loss: 0.0249412511578218 | Epoch: 236 | MeanAbsolutel

MeanAbsoluteError: 0.1246964409947395 | Loss: 0.0255242762161652 | Epoch: 238 |
```

```
MeanAbsoluteError: 0.1235716864466667 | Loss: 0.0246927623393276 | Epoch: 239 | MeanAbsoluteE

MeanAbsoluteError: 0.1246307343244553 | Loss: 0.0250004822772462 | Epoch: 241 |

MeanAbsoluteError: 0.1244416162371635 | Loss: 0.0249699609160113 | Epoch: 242 | MeanAbsoluteE

MeanAbsoluteError: 0.1203196048736572 | Loss: 0.0247471561606805 | Epoch: 244 |

MeanAbsoluteError: 0.1184556931257248 | Loss: 0.0229529287018522 | Epoch: 245 | MeanAbsoluteE

MeanAbsoluteError: 0.1213295608758926 | Loss: 0.0253280114775165 | Epoch: 247 |

MeanAbsoluteError: 0.1227372363209724 | Loss: 0.0254023163742750 | Epoch: 248 | MeanAbsoluteE

MeanAbsoluteError: 0.1261341869831085 | Loss: 0.0255179818058969 | Epoch: 250 |

MeanAbsoluteError: 0.1252125203609467 | Loss: 0.0254568494171690 | Epoch: 251 | MeanAbsoluteE

MeanAbsoluteError: 0.1285345703363419 | Loss: 0.0254357593695264 | Epoch: 253 |

MeanAbsoluteError: 0.1182083636522293 | Loss: 0.0229149588735648 | Epoch: 254 | MeanAbsoluteE

MeanAbsoluteError: 0.1267596632242203 | Loss: 0.0255098177682764 | Epoch: 256 |

MeanAbsoluteError: 0.1243192404508591 | Loss: 0.0250096383787362 | Epoch: 257 | MeanAbsoluteE

MeanAbsoluteError: 0.1217936947941780 | Loss: 0.0242561478248657 | Epoch: 259 |

MeanAbsoluteError: 0.1200099810957909 | Loss: 0.0231619556551595 | Epoch: 260 | MeanAbsoluteE

MeanAbsoluteError: 0.1241989806294441 | Loss: 0.0249710375499368 | Epoch: 262 |

MeanAbsoluteError: 0.1270198822021484 | Loss: 0.0256064807792427 | Epoch: 263 | MeanAbsoluteE

MeanAbsoluteError: 0.1217404454946518 | Loss: 0.0230496524062861 | Epoch: 265 |

MeanAbsoluteError: 0.1195053458213806 | Loss: 0.0236737859742910 | Epoch: 266 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.1190129071474075 | Loss: 0.0238029774913108 | Epoch: 268 |

MeanAbsoluteError: 0.1203722283244133 | Loss: 0.0252195619464813 | Epoch: 269 | MeanAbsolutel

MeanAbsoluteError: 0.1229190155863762 | Loss: 0.0238524518819759 | Epoch: 271 |

MeanAbsoluteError: 0.1250833123922348 | Loss: 0.0263598888979444 | Epoch: 272 | MeanAbsolutel

MeanAbsoluteError: 0.1175298541784286 | Loss: 0.0222856664129843 | Epoch: 274 |

MeanAbsoluteError: 0.1215374767780304 | Loss: 0.0239659193527526 | Epoch: 275 | MeanAbsolutel

MeanAbsoluteError: 0.1204968392848969 | Loss: 0.0238031015742066 | Epoch: 277 |

MeanAbsoluteError: 0.1142812967300415 | Loss: 0.0217369371368841 | Epoch: 278 | MeanAbsolutel

MeanAbsoluteError: 0.1208290383219719 | Loss: 0.0242415063893228 | Epoch: 280 |

MeanAbsoluteError: 0.1215936020016670 | Loss: 0.0242063757866466 | Epoch: 281 | MeanAbsolutel

MeanAbsoluteError: 0.1198621466755867 | Loss: 0.0229228491831843 | Epoch: 283 |

MeanAbsoluteError: 0.1194350570440292 | Loss: 0.0233001606915301 | Epoch: 284 | MeanAbsolutel

MeanAbsoluteError: 0.1273110657930374 | Loss: 0.0256627194003280 | Epoch: 286 |

MeanAbsoluteError: 0.1236405819654465 | Loss: 0.0251977036071791 | Epoch: 287 | MeanAbsolutel

MeanAbsoluteError: 0.1218986362218857 | Loss: 0.0245290178714519 | Epoch: 289 |

MeanAbsoluteError: 0.1177783459424973 | Loss: 0.0224440150225807 | Epoch: 290 | MeanAbsolutel

MeanAbsoluteError: 0.1236680820584297 | Loss: 0.0256997018217226 | Epoch: 292 |

MeanAbsoluteError: 0.1135459765791893 | Loss: 0.0223021862072589 | Epoch: 293 | MeanAbsolutel

MeanAbsoluteError: 0.1225657239556313 | Loss: 0.0243479311783449 | Epoch: 295 |
```

```
MeanAbsoluteError: 0.1164296120405197 | Loss: 0.0228280045588811 | Epoch: 296 | MeanAbsoluteE

MeanAbsoluteError: 0.1189049780368805 | Loss: 0.0224264792558279 | Epoch: 298 |

MeanAbsoluteError: 0.1118806824088097 | Loss: 0.0210086449031951 | Epoch: 299 | MeanAbsoluteE

MeanAbsoluteError: 0.1198000609874725 | Loss: 0.0241238151715515 | Epoch: 301 |

MeanAbsoluteError: 0.1191958338022232 | Loss: 0.0229979263315909 | Epoch: 302 | MeanAbsoluteE

MeanAbsoluteError: 0.1142625808715820 | Loss: 0.0223172860063823 | Epoch: 304 |

MeanAbsoluteError: 0.1267854124307632 | Loss: 0.0255951875886725 | Epoch: 305 | MeanAbsoluteE

MeanAbsoluteError: 0.1160637438297272 | Loss: 0.0219001621965193 | Epoch: 307 |

MeanAbsoluteError: 0.1254428029060364 | Loss: 0.0246356707563003 | Epoch: 308 | MeanAbsoluteE

MeanAbsoluteError: 0.1198697388172150 | Loss: 0.0240270768937868 | Epoch: 310 |

MeanAbsoluteError: 0.1225919499993324 | Loss: 0.0236331517091215 | Epoch: 311 | MeanAbsoluteE

MeanAbsoluteError: 0.1198123544454575 | Loss: 0.0244060188981045 | Epoch: 313 |

MeanAbsoluteError: 0.1143840923905373 | Loss: 0.0210581055318714 | Epoch: 314 | MeanAbsoluteE

MeanAbsoluteError: 0.1225596591830254 | Loss: 0.0237894396977208 | Epoch: 316 |

MeanAbsoluteError: 0.1163892596960068 | Loss: 0.0229394549494949 | Epoch: 317 | MeanAbsoluteE

MeanAbsoluteError: 0.1143676638603210 | Loss: 0.0219968613270127 | Epoch: 319 |

MeanAbsoluteError: 0.1215145513415337 | Loss: 0.0245567772169791 | Epoch: 320 | MeanAbsoluteE

MeanAbsoluteError: 0.1194207668304443 | Loss: 0.0231373055545009 | Epoch: 322 |

MeanAbsoluteError: 0.1196876019239426 | Loss: 0.0234859224697963 | Epoch: 323 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.1193092837929726 | Loss: 0.0219882492623583 | Epoch: 325 |
MeanAbsoluteError: 0.1184501796960831 | Loss: 0.0221367123752134 | Epoch: 326 | MeanAbsolutel
MeanAbsoluteError: 0.1216489151120186 | Loss: 0.0239730344263565 | Epoch: 328 |
MeanAbsoluteError: 0.1228993088006973 | Loss: 0.0247531017203316 | Epoch: 329 | MeanAbsolutel
MeanAbsoluteError: 0.1245453283190727 | Loss: 0.0239852907053137 | Epoch: 331 |
MeanAbsoluteError: 0.1224043890833855 | Loss: 0.0240143576140205 | Epoch: 332 | MeanAbsolutel
MeanAbsoluteError: 0.1185982003808022 | Loss: 0.0221785883077731 | Epoch: 334 |
MeanAbsoluteError: 0.1111389696598053 | Loss: 0.0219761552828519 | Epoch: 335 | MeanAbsolutel
MeanAbsoluteError: 0.1166688129305840 | Loss: 0.0222624457739100 | Epoch: 337 |
MeanAbsoluteError: 0.1201581954956055 | Loss: 0.0233182413281368 | Epoch: 338 | MeanAbsolutel
MeanAbsoluteError: 0.1180995553731918 | Loss: 0.0227548580358659 | Epoch: 340 |
MeanAbsoluteError: 0.1149417236447334 | Loss: 0.0227123566926457 | Epoch: 341 | MeanAbsolutel
MeanAbsoluteError: 0.1174345016479492 | Loss: 0.0220979495479454 | Epoch: 343 |
MeanAbsoluteError: 0.1221868097782135 | Loss: 0.0236479326964278 | Epoch: 344 | MeanAbsolutel
MeanAbsoluteError: 0.1151180788874626 | Loss: 0.0221483386929807 | Epoch: 346 |
MeanAbsoluteError: 0.1195265576243401 | Loss: 0.0232190396747319 | Epoch: 347 | MeanAbsolutel
MeanAbsoluteError: 0.1131289899349213 | Loss: 0.0212830391214811 | Epoch: 349 |
MeanAbsoluteError: 0.1130820959806442 | Loss: 0.0202506073996968 | Epoch: 350 | MeanAbsolutel
MeanAbsoluteError: 0.1192318201065063 | Loss: 0.0235490064306456 | Epoch: 352 |
```

```
MeanAbsoluteError: 0.1179587393999100 | Loss: 0.0226025096927575 | Epoch: 353 | MeanAbsoluteE

MeanAbsoluteError: 0.1131815090775490 | Loss: 0.0211008325383106 | Epoch: 355 |

MeanAbsoluteError: 0.1161853894591331 | Loss: 0.0220053671086377 | Epoch: 356 | MeanAbsoluteE

MeanAbsoluteError: 0.1184464246034622 | Loss: 0.0228515797306803 | Epoch: 358 |

MeanAbsoluteError: 0.1106319427490234 | Loss: 0.0202156037218325 | Epoch: 359 | MeanAbsoluteE

MeanAbsoluteError: 0.1126907989382744 | Loss: 0.0211005326734933 | Epoch: 361 |

MeanAbsoluteError: 0.1158623769879341 | Loss: 0.0218977348586001 | Epoch: 362 | MeanAbsoluteE

MeanAbsoluteError: 0.1170916333794594 | Loss: 0.0225724309059054 | Epoch: 364 |

MeanAbsoluteError: 0.1246641799807549 | Loss: 0.0245320785691729 | Epoch: 365 | MeanAbsoluteE

MeanAbsoluteError: 0.1210922002792358 | Loss: 0.0229833814828695 | Epoch: 367 |

MeanAbsoluteError: 0.1161371991038322 | Loss: 0.0221753664068334 | Epoch: 368 | MeanAbsoluteE

MeanAbsoluteError: 0.1175065264105797 | Loss: 0.0226643902317543 | Epoch: 370 |

MeanAbsoluteError: 0.1209406256675720 | Loss: 0.0228025892867784 | Epoch: 371 | MeanAbsoluteE

MeanAbsoluteError: 0.1156608089804649 | Loss: 0.0213373158314910 | Epoch: 373 |

MeanAbsoluteError: 0.1153079196810722 | Loss: 0.0220805951141908 | Epoch: 374 | MeanAbsoluteE

MeanAbsoluteError: 0.1171360760927200 | Loss: 0.0218527428417246 | Epoch: 376 |

MeanAbsoluteError: 0.1090600490570068 | Loss: 0.0193684554585100 | Epoch: 377 | MeanAbsoluteE

MeanAbsoluteError: 0.1138937026262283 | Loss: 0.0212620969795777 | Epoch: 379 |

MeanAbsoluteError: 0.1122894138097763 | Loss: 0.0214363883701541 | Epoch: 380 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.1159460544586182 | Loss: 0.0214616101143959 | Epoch: 382 |

MeanAbsoluteError: 0.1130133420228958 | Loss: 0.0219188302281267 | Epoch: 383 | MeanAbsolutel

MeanAbsoluteError: 0.1132514700293541 | Loss: 0.0213264690758660 | Epoch: 385 |

MeanAbsoluteError: 0.1130205169320107 | Loss: 0.0210562960088525 | Epoch: 386 | MeanAbsolutel

MeanAbsoluteError: 0.1163441240787506 | Loss: 0.0222871479072698 | Epoch: 388 |

MeanAbsoluteError: 0.1088204979896545 | Loss: 0.0198925708839670 | Epoch: 389 | MeanAbsolutel

MeanAbsoluteError: 0.1120014712214470 | Loss: 0.0200473753754826 | Epoch: 391 |

MeanAbsoluteError: 0.1085745021700859 | Loss: 0.0198431442726481 | Epoch: 392 | MeanAbsolutel

MeanAbsoluteError: 0.1143083199858665 | Loss: 0.0214385439609881 | Epoch: 394 |

MeanAbsoluteError: 0.1037482470273972 | Loss: 0.0176697835202989 | Epoch: 395 | MeanAbsolutel

MeanAbsoluteError: 0.1177421808242798 | Loss: 0.0228064203822881 | Epoch: 397 |

MeanAbsoluteError: 0.1145924031734467 | Loss: 0.0211820441262535 | Epoch: 398 | MeanAbsolutel

MeanAbsoluteError: 0.1175473257899284 | Loss: 0.0223289670939266 | Epoch: 400 |

MeanAbsoluteError: 0.1140572875738144 | Loss: 0.0207435708323222 | Epoch: 401 | MeanAbsolutel

MeanAbsoluteError: 0.1161898151040077 | Loss: 0.0222632164807874 | Epoch: 403 |

MeanAbsoluteError: 0.1101500168442726 | Loss: 0.0192168896140841 | Epoch: 404 | MeanAbsolutel

MeanAbsoluteError: 0.1067449823021889 | Loss: 0.0182773087769359 | Epoch: 406 |

MeanAbsoluteError: 0.1146385818719864 | Loss: 0.0211621506093070 | Epoch: 407 | MeanAbsolutel

MeanAbsoluteError: 0.1127190366387367 | Loss: 0.0210785393838887 | Epoch: 409 |
```

```
MeanAbsoluteError: 0.1107881516218185 | Loss: 0.0194759002459856 | Epoch: 410 | MeanAbsoluteE

MeanAbsoluteError: 0.1155132651329041 | Loss: 0.0206013054512732 | Epoch: 412 |

MeanAbsoluteError: 0.1174905002117157 | Loss: 0.0219221429825605 | Epoch: 413 | MeanAbsoluteE

MeanAbsoluteError: 0.1061775684356689 | Loss: 0.0196146224048562 | Epoch: 415 |

MeanAbsoluteError: 0.1106395497918129 | Loss: 0.0203149637357274 | Epoch: 416 | MeanAbsoluteE

MeanAbsoluteError: 0.1033112853765488 | Loss: 0.0177301753307014 | Epoch: 418 |

MeanAbsoluteError: 0.1139743402600288 | Loss: 0.0214815212661051 | Epoch: 419 | MeanAbsoluteE

MeanAbsoluteError: 0.1099766939878464 | Loss: 0.0204593109873531 | Epoch: 421 |

MeanAbsoluteError: 0.1106279790401459 | Loss: 0.0202868603652072 | Epoch: 422 | MeanAbsoluteE

MeanAbsoluteError: 0.1126058995723724 | Loss: 0.0193337585616003 | Epoch: 424 |

MeanAbsoluteError: 0.1158485934138298 | Loss: 0.0221099036857292 | Epoch: 425 | MeanAbsoluteE

MeanAbsoluteError: 0.1088843792676926 | Loss: 0.0213150805029242 | Epoch: 427 |

MeanAbsoluteError: 0.1116317212581635 | Loss: 0.0205813969495648 | Epoch: 428 | MeanAbsoluteE

MeanAbsoluteError: 0.1187642440199852 | Loss: 0.0233737901117517 | Epoch: 430 |

MeanAbsoluteError: 0.1162345781922340 | Loss: 0.0217997111051955 | Epoch: 431 | MeanAbsoluteE

MeanAbsoluteError: 0.1129278540611267 | Loss: 0.0207383612122309 | Epoch: 433 |

MeanAbsoluteError: 0.1111578494310379 | Loss: 0.0194372346659717 | Epoch: 434 | MeanAbsoluteE

MeanAbsoluteError: 0.1105150505900383 | Loss: 0.0199027793501349 | Epoch: 436 |

MeanAbsoluteError: 0.1003053039312363 | Loss: 0.0179786354268920 | Epoch: 437 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.1165917813777924 | Loss: 0.0221960456457358 | Epoch: 439 |

MeanAbsoluteError: 0.1090754196047783 | Loss: 0.0194519754031595 | Epoch: 440 | MeanAbsolutel

MeanAbsoluteError: 0.1122975945472717 | Loss: 0.0206569075395964 | Epoch: 442 |

MeanAbsoluteError: 0.1061591506004333 | Loss: 0.0185948924349941 | Epoch: 443 | MeanAbsolutel

MeanAbsoluteError: 0.1117988973855972 | Loss: 0.0207606756884585 | Epoch: 445 |

MeanAbsoluteError: 0.1129675805568695 | Loss: 0.0200058237829793 | Epoch: 446 | MeanAbsolutel

MeanAbsoluteError: 0.1098176985979080 | Loss: 0.0189633972918576 | Epoch: 448 |

MeanAbsoluteError: 0.1128693372011185 | Loss: 0.0212382517832642 | Epoch: 449 | MeanAbsolutel

MeanAbsoluteError: 0.1070020198822021 | Loss: 0.0192109969803035 | Epoch: 451 |

MeanAbsoluteError: 0.1094160079956055 | Loss: 0.0201406416787480 | Epoch: 452 | MeanAbsolutel

MeanAbsoluteError: 0.1110012084245682 | Loss: 0.0199683087851493 | Epoch: 454 |

MeanAbsoluteError: 0.1022091582417488 | Loss: 0.0177301865612875 | Epoch: 455 | MeanAbsolutel

MeanAbsoluteError: 0.1129143387079239 | Loss: 0.0209964807648781 | Epoch: 457 |

MeanAbsoluteError: 0.1112044751644135 | Loss: 0.0192987068924413 | Epoch: 458 | MeanAbsolutel

MeanAbsoluteError: 0.1112016513943672 | Loss: 0.0208493125126309 | Epoch: 460 |

MeanAbsoluteError: 0.1130627319216728 | Loss: 0.0203245671900610 | Epoch: 461 | MeanAbsolutel

MeanAbsoluteError: 0.1021970734000206 | Loss: 0.0166113768458308 | Epoch: 463 |

MeanAbsoluteError: 0.1038033142685890 | Loss: 0.0185865639350959 | Epoch: 464 | MeanAbsolutel

MeanAbsoluteError: 0.1140823140740395 | Loss: 0.0214876110704305 | Epoch: 466 |
```

```
MeanAbsoluteError: 0.1088520064949989 | Loss: 0.0188597057857745 | Epoch: 467 | MeanAbsoluteE

MeanAbsoluteError: 0.1089070513844490 | Loss: 0.0196191236588250 | Epoch: 469 |

MeanAbsoluteError: 0.1031892523169518 | Loss: 0.0170571907673730 | Epoch: 470 | MeanAbsoluteE

MeanAbsoluteError: 0.1092367172241211 | Loss: 0.0199081198136749 | Epoch: 472 |

MeanAbsoluteError: 0.1082919538021088 | Loss: 0.0188992976430745 | Epoch: 473 | MeanAbsoluteE

MeanAbsoluteError: 0.1105509698390961 | Loss: 0.0196738283910478 | Epoch: 475 |

MeanAbsoluteError: 0.1007082015275955 | Loss: 0.0172206508181686 | Epoch: 476 | MeanAbsoluteE

MeanAbsoluteError: 0.1138883605599403 | Loss: 0.0213356835437783 | Epoch: 478 |

MeanAbsoluteError: 0.1053787767887115 | Loss: 0.0183389637698807 | Epoch: 479 | MeanAbsoluteE

MeanAbsoluteError: 0.1075319424271584 | Loss: 0.0193912296973576 | Epoch: 481 |

MeanAbsoluteError: 0.1085772961378098 | Loss: 0.0184928590581209 | Epoch: 482 | MeanAbsoluteE

MeanAbsoluteError: 0.1058026924729347 | Loss: 0.0183967071293470 | Epoch: 484 |

MeanAbsoluteError: 0.1033527404069901 | Loss: 0.0185719964856495 | Epoch: 485 | MeanAbsoluteE

MeanAbsoluteError: 0.1041722744703293 | Loss: 0.0188757271665721 | Epoch: 487 |

MeanAbsoluteError: 0.1073627471923828 | Loss: 0.0190380018253078 | Epoch: 488 | MeanAbsoluteE

MeanAbsoluteError: 0.1095566526055336 | Loss: 0.0199752556732468 | Epoch: 490 |

MeanAbsoluteError: 0.1133541092276573 | Loss: 0.0211002807786766 | Epoch: 491 | MeanAbsoluteE

MeanAbsoluteError: 0.1076906844973564 | Loss: 0.0186487314621627 | Epoch: 493 |

MeanAbsoluteError: 0.1026776432991028 | Loss: 0.0169396603833108 | Epoch: 494 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.1074662655591965 | Loss: 0.0199592913883680 | Epoch: 496 |

MeanAbsoluteError: 0.1070212945342064 | Loss: 0.0185593779266977 | Epoch: 497 | MeanAbsolutel

MeanAbsoluteError: 0.1068575680255890 | Loss: 0.0183447640949150 | Epoch: 499 |

MeanAbsoluteError: 0.1043617948889732 | Loss: 0.0174353746610480 | Epoch: 500 | MeanAbsolutel

MeanAbsoluteError: 0.1091047450900078 | Loss: 0.0194571588205872 | Epoch: 502 |

MeanAbsoluteError: 0.1044304668903351 | Loss: 0.0179444825119572 | Epoch: 503 | MeanAbsolutel

MeanAbsoluteError: 0.1031957492232323 | Loss: 0.0192779188084144 | Epoch: 505 |

MeanAbsoluteError: 0.1073712930083275 | Loss: 0.0189493819124376 | Epoch: 506 | MeanAbsolutel

MeanAbsoluteError: 0.0991007685661316 | Loss: 0.0163587164085523 | Epoch: 508 |

MeanAbsoluteError: 0.0975435674190521 | Loss: 0.0165684791925014 | Epoch: 509 | MeanAbsolutel

MeanAbsoluteError: 0.1153217330574989 | Loss: 0.0224251791775411 | Epoch: 511 |

MeanAbsoluteError: 0.1040936857461929 | Loss: 0.0182242198821041 | Epoch: 512 | MeanAbsolutel

MeanAbsoluteError: 0.1076111346483231 | Loss: 0.0189668997033732 | Epoch: 514 |

MeanAbsoluteError: 0.1053425520658493 | Loss: 0.0182589051259371 | Epoch: 515 | MeanAbsolutel

MeanAbsoluteError: 0.1034887358546257 | Loss: 0.0179697462833064 | Epoch: 517 |

MeanAbsoluteError: 0.1034315079450607 | Loss: 0.0176187906923587 | Epoch: 518 | MeanAbsolutel

MeanAbsoluteError: 0.1031467318534851 | Loss: 0.0177417629333892 | Epoch: 520 |

MeanAbsoluteError: 0.1067584455013275 | Loss: 0.0178791456364949 | Epoch: 521 | MeanAbsolutel

MeanAbsoluteError: 0.1069447323679924 | Loss: 0.0197612547214764 | Epoch: 523 |
```

```
MeanAbsoluteError: 0.1012611985206604 | Loss: 0.0183841567702014 | Epoch: 524 | MeanAbsoluteE

MeanAbsoluteError: 0.1014211028814316 | Loss: 0.0171619890687483 | Epoch: 526 |

MeanAbsoluteError: 0.1022388264536858 | Loss: 0.0177298965651547 | Epoch: 527 | MeanAbsoluteE

MeanAbsoluteError: 0.1003714278340340 | Loss: 0.0167494728337291 | Epoch: 529 |

MeanAbsoluteError: 0.1014671623706818 | Loss: 0.0180910395799583 | Epoch: 530 | MeanAbsoluteE

MeanAbsoluteError: 0.1012425348162651 | Loss: 0.0168570845215193 | Epoch: 532 |

MeanAbsoluteError: 0.1043267026543617 | Loss: 0.0185976531713095 | Epoch: 533 | MeanAbsoluteE

MeanAbsoluteError: 0.1005688905715942 | Loss: 0.0169889893371146 | Epoch: 535 |

MeanAbsoluteError: 0.1066789254546165 | Loss: 0.0191238019492933 | Epoch: 536 | MeanAbsoluteE

MeanAbsoluteError: 0.0961455851793289 | Loss: 0.0154500663136181 | Epoch: 538 |

MeanAbsoluteError: 0.1072838976979256 | Loss: 0.0182510942290052 | Epoch: 539 | MeanAbsoluteE

MeanAbsoluteError: 0.1032533869147301 | Loss: 0.0177926225282257 | Epoch: 541 |

MeanAbsoluteError: 0.1030658408999443 | Loss: 0.0173122073818134 | Epoch: 542 | MeanAbsoluteE

MeanAbsoluteError: 0.1104598492383957 | Loss: 0.0200687033577318 | Epoch: 544 |

MeanAbsoluteError: 0.1032923981547356 | Loss: 0.0182967016598802 | Epoch: 545 | MeanAbsoluteE

MeanAbsoluteError: 0.1052022278308868 | Loss: 0.0180525866113506 | Epoch: 547 |

MeanAbsoluteError: 0.1112811639904976 | Loss: 0.0198351004467501 | Epoch: 548 | MeanAbsoluteE

MeanAbsoluteError: 0.0957020446658134 | Loss: 0.0157121419538695 | Epoch: 550 |

MeanAbsoluteError: 0.0956940278410912 | Loss: 0.0154285403405083 | Epoch: 551 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.1016877219080925 | Loss: 0.0171927248193242 | Epoch: 553 |

MeanAbsoluteError: 0.1013165935873985 | Loss: 0.0169122306129187 | Epoch: 554 | MeanAbsolutel

MeanAbsoluteError: 0.1012064516544342 | Loss: 0.0173696111564762 | Epoch: 556 |

MeanAbsoluteError: 0.1048536673188210 | Loss: 0.0182420898822117 | Epoch: 557 | MeanAbsolutel

MeanAbsoluteError: 0.0976944267749786 | Loss: 0.0172976162273941 | Epoch: 559 |

MeanAbsoluteError: 0.0976610481739044 | Loss: 0.0161553288437790 | Epoch: 560 | MeanAbsolutel

MeanAbsoluteError: 0.1083266586065292 | Loss: 0.0181218652070190 | Epoch: 562 |

MeanAbsoluteError: 0.1049693897366524 | Loss: 0.0184889280837918 | Epoch: 563 | MeanAbsolutel

MeanAbsoluteError: 0.1008365005254745 | Loss: 0.0172986108246065 | Epoch: 565 |

MeanAbsoluteError: 0.0969493463635445 | Loss: 0.0161678165938065 | Epoch: 566 | MeanAbsolutel

MeanAbsoluteError: 0.1026474162936211 | Loss: 0.0171772614606743 | Epoch: 568 |

MeanAbsoluteError: 0.0986035838723183 | Loss: 0.0161479429431589 | Epoch: 569 | MeanAbsolutel

MeanAbsoluteError: 0.0976569429039955 | Loss: 0.0157769067530899 | Epoch: 571 |

MeanAbsoluteError: 0.0989868417382240 | Loss: 0.0170392362462007 | Epoch: 572 | MeanAbsolutel

MeanAbsoluteError: 0.1007998809218407 | Loss: 0.0175795994324593 | Epoch: 574 |

MeanAbsoluteError: 0.1000328510999680 | Loss: 0.0168671787200825 | Epoch: 575 | MeanAbsolutel

MeanAbsoluteError: 0.1022880151867867 | Loss: 0.0175176319325207 | Epoch: 577 |

MeanAbsoluteError: 0.0957431048154831 | Loss: 0.0149461985646485 | Epoch: 578 | MeanAbsolutel

MeanAbsoluteError: 0.1032386496663094 | Loss: 0.0174026367987972 | Epoch: 580 |
```

```
MeanAbsoluteError: 0.0983982384204865 | Loss: 0.0166201997429016 | Epoch: 581 | MeanAbsoluteE

MeanAbsoluteError: 0.0938294678926468 | Loss: 0.0152162210562043 | Epoch: 583 |

MeanAbsoluteError: 0.0961143076419830 | Loss: 0.0162344349373598 | Epoch: 584 | MeanAbsoluteE

MeanAbsoluteError: 0.0989495068788528 | Loss: 0.0165485622493240 | Epoch: 586 |

MeanAbsoluteError: 0.1037187129259109 | Loss: 0.0175498040694523 | Epoch: 587 | MeanAbsoluteE

MeanAbsoluteError: 0.0930963680148125 | Loss: 0.0145389358918571 | Epoch: 589 |

MeanAbsoluteError: 0.1110416948795319 | Loss: 0.0200054403899412 | Epoch: 590 | MeanAbsoluteE

MeanAbsoluteError: 0.1029701232910156 | Loss: 0.0170861399539475 | Epoch: 592 |

MeanAbsoluteError: 0.0991387888789177 | Loss: 0.0160304785400513 | Epoch: 593 | MeanAbsoluteE

MeanAbsoluteError: 0.0970365479588509 | Loss: 0.0158517592945524 | Epoch: 595 |

MeanAbsoluteError: 0.0975929945707321 | Loss: 0.0165572154321975 | Epoch: 596 | MeanAbsoluteE

MeanAbsoluteError: 0.1032423079013824 | Loss: 0.0178222090355121 | Epoch: 598 |

MeanAbsoluteError: 0.0955274105072021 | Loss: 0.0154074033629149 | Epoch: 599 | MeanAbsoluteE

MeanAbsoluteError: 0.0937800407409668 | Loss: 0.0156470010548461 | Epoch: 601 |

MeanAbsoluteError: 0.1012295410037041 | Loss: 0.0168281149937926 | Epoch: 602 | MeanAbsoluteE

MeanAbsoluteError: 0.1009463220834732 | Loss: 0.0180626184558484 | Epoch: 604 |

MeanAbsoluteError: 0.0947722047567368 | Loss: 0.0149160144079360 | Epoch: 605 | MeanAbsoluteE

MeanAbsoluteError: 0.0942825600504875 | Loss: 0.0151703653940785 | Epoch: 607 |

MeanAbsoluteError: 0.1015526875853539 | Loss: 0.0177609964968966 | Epoch: 608 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.1063989028334618 | Loss: 0.0175637328504672 | Epoch: 610 |

MeanAbsoluteError: 0.0991244316101074 | Loss: 0.0166514532416477 | Epoch: 611 | MeanAbsoluteE

MeanAbsoluteError: 0.0967791527509689 | Loss: 0.0146798651420977 | Epoch: 613 |

MeanAbsoluteError: 0.0974676385521889 | Loss: 0.0166920117833070 | Epoch: 614 | MeanAbsoluteE

MeanAbsoluteError: 0.0994392782449722 | Loss: 0.0168958605242854 | Epoch: 616 |

MeanAbsoluteError: 0.1009593382477760 | Loss: 0.0169848190987250 | Epoch: 617 | MeanAbsoluteE

MeanAbsoluteError: 0.0967506691813469 | Loss: 0.0152882742330015 | Epoch: 619 |

MeanAbsoluteError: 0.0949692428112030 | Loss: 0.0151178884057238 | Epoch: 620 | MeanAbsoluteE

MeanAbsoluteError: 0.0944637730717659 | Loss: 0.0151726977157038 | Epoch: 622 |

MeanAbsoluteError: 0.0982013046741486 | Loss: 0.0156392288952581 | Epoch: 623 | MeanAbsoluteE

MeanAbsoluteError: 0.0908328741788864 | Loss: 0.0151709146050901 | Epoch: 625 |

MeanAbsoluteError: 0.0927953347563744 | Loss: 0.0142623809368039 | Epoch: 626 | MeanAbsoluteE

MeanAbsoluteError: 0.0953974500298500 | Loss: 0.0157236374068695 | Epoch: 628 |

MeanAbsoluteError: 0.0972630232572556 | Loss: 0.0161563555866693 | Epoch: 629 | MeanAbsoluteE

MeanAbsoluteError: 0.0952812135219574 | Loss: 0.0149958149523203 | Epoch: 631 |

MeanAbsoluteError: 0.0956340953707695 | Loss: 0.0159929800519603 | Epoch: 632 | MeanAbsoluteE

MeanAbsoluteError: 0.0916225016117096 | Loss: 0.0143741240124170 | Epoch: 634 |

MeanAbsoluteError: 0.0926685929298401 | Loss: 0.0150796643912812 | Epoch: 635 | MeanAbsoluteE

MeanAbsoluteError: 0.0903482660651207 | Loss: 0.0133340552879721 | Epoch: 637 |
```

```
MeanAbsoluteError: 0.0924038141965866 | Loss: 0.0140900135477083 | Epoch: 638 | MeanAbsoluteE

MeanAbsoluteError: 0.0913464948534966 | Loss: 0.0143619613607007 | Epoch: 640 |

MeanAbsoluteError: 0.0942121669650078 | Loss: 0.0146234798495182 | Epoch: 641 | MeanAbsoluteE

MeanAbsoluteError: 0.0960903838276863 | Loss: 0.0158145495447631 | Epoch: 643 |

MeanAbsoluteError: 0.0994423255324364 | Loss: 0.0159278475097381 | Epoch: 644 | MeanAbsoluteE

MeanAbsoluteError: 0.0953540578484535 | Loss: 0.0152846255711969 | Epoch: 646 |

MeanAbsoluteError: 0.0955077633261681 | Loss: 0.0154337800083158 | Epoch: 647 | MeanAbsoluteE

MeanAbsoluteError: 0.0933151915669441 | Loss: 0.0150365700113374 | Epoch: 649 |

MeanAbsoluteError: 0.0929899364709854 | Loss: 0.0141515090686153 | Epoch: 650 | MeanAbsoluteE

MeanAbsoluteError: 0.0943574011325836 | Loss: 0.0144483166945095 | Epoch: 652 |

MeanAbsoluteError: 0.0982109680771828 | Loss: 0.0158709337642843 | Epoch: 653 | MeanAbsoluteE

MeanAbsoluteError: 0.0965842530131340 | Loss: 0.0161847001681842 | Epoch: 655 |

MeanAbsoluteError: 0.0961767807602882 | Loss: 0.0154472003212747 | Epoch: 656 | MeanAbsoluteE

MeanAbsoluteError: 0.0914197564125061 | Loss: 0.0135365715262985 | Epoch: 658 |

MeanAbsoluteError: 0.0965374112129211 | Loss: 0.0161410397714887 | Epoch: 659 | MeanAbsoluteE

MeanAbsoluteError: 0.0862118229269981 | Loss: 0.0124753704971711 | Epoch: 661 |

MeanAbsoluteError: 0.0941973328590393 | Loss: 0.0143871776682984 | Epoch: 662 | MeanAbsoluteE

MeanAbsoluteError: 0.0924053490161896 | Loss: 0.0138245465293585 | Epoch: 664 |

MeanAbsoluteError: 0.0943065360188484 | Loss: 0.0147915934391009 | Epoch: 665 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.0877766385674477 | Loss: 0.0134749899315648 | Epoch: 667 |

MeanAbsoluteError: 0.0891092717647552 | Loss: 0.0136814642985034 | Epoch: 668 | MeanAbsolutel

MeanAbsoluteError: 0.1001598984003067 | Loss: 0.0175169296632521 | Epoch: 670 |

MeanAbsoluteError: 0.0956887081265450 | Loss: 0.0148739303835221 | Epoch: 671 | MeanAbsolutel

MeanAbsoluteError: 0.0898870229721069 | Loss: 0.0136956985358969 | Epoch: 673 |

MeanAbsoluteError: 0.0930449515581131 | Loss: 0.0147369063267494 | Epoch: 674 | MeanAbsolutel

MeanAbsoluteError: 0.0928883254528046 | Loss: 0.0151533641509983 | Epoch: 676 |

MeanAbsoluteError: 0.0957888439297676 | Loss: 0.0150557202855028 | Epoch: 677 | MeanAbsolutel

MeanAbsoluteError: 0.0948926359415054 | Loss: 0.0151187013328308 | Epoch: 679 |

MeanAbsoluteError: 0.0901934653520584 | Loss: 0.0140082426281394 | Epoch: 680 | MeanAbsolutel

MeanAbsoluteError: 0.0884562432765961 | Loss: 0.0129099848298938 | Epoch: 682 |

MeanAbsoluteError: 0.0934609100222588 | Loss: 0.0142482270121400 | Epoch: 683 | MeanAbsolutel

MeanAbsoluteError: 0.0906330570578575 | Loss: 0.0143446560424248 | Epoch: 685 |

MeanAbsoluteError: 0.0903934761881828 | Loss: 0.0141147623146147 | Epoch: 686 | MeanAbsolutel

MeanAbsoluteError: 0.0943584963679314 | Loss: 0.0151742838945252 | Epoch: 688 |

MeanAbsoluteError: 0.0969038009643555 | Loss: 0.0156067289850519 | Epoch: 689 | MeanAbsolutel

MeanAbsoluteError: 0.0930010676383972 | Loss: 0.0149562277570658 | Epoch: 691 |

MeanAbsoluteError: 0.0915986746549606 | Loss: 0.0159623541114464 | Epoch: 692 | MeanAbsolutel

MeanAbsoluteError: 0.0921640172600746 | Loss: 0.0149641823869752 | Epoch: 694 |
```

```
MeanAbsoluteError: 0.0917146876454353 | Loss: 0.0149230020907028 | Epoch: 695 | MeanAbsoluteE

MeanAbsoluteError: 0.0892915502190590 | Loss: 0.0134321752321557 | Epoch: 697 |

MeanAbsoluteError: 0.0912369936704636 | Loss: 0.0137544005184706 | Epoch: 698 | MeanAbsoluteE

MeanAbsoluteError: 0.0915101766586304 | Loss: 0.0148272872219483 | Epoch: 700 |

MeanAbsoluteError: 0.0901134163141251 | Loss: 0.0139484527889969 | Epoch: 701 | MeanAbsoluteE

MeanAbsoluteError: 0.0866718515753746 | Loss: 0.0138874019054007 | Epoch: 703 |

MeanAbsoluteError: 0.0911324024200439 | Loss: 0.0138184867255040 | Epoch: 704 | MeanAbsoluteE

MeanAbsoluteError: 0.0937384441494942 | Loss: 0.0147017357690493 | Epoch: 706 |

MeanAbsoluteError: 0.0945695340633392 | Loss: 0.0152396774454974 | Epoch: 707 | MeanAbsoluteE

MeanAbsoluteError: 0.0860642194747925 | Loss: 0.0128454439997343 | Epoch: 709 |

MeanAbsoluteError: 0.0913770496845245 | Loss: 0.0134927070279567 | Epoch: 710 | MeanAbsoluteE

MeanAbsoluteError: 0.0939768478274345 | Loss: 0.0147362866539819 | Epoch: 712 |

MeanAbsoluteError: 0.0967347919940948 | Loss: 0.0159946287120025 | Epoch: 713 | MeanAbsoluteE

MeanAbsoluteError: 0.0907365828752518 | Loss: 0.0137782287185109 | Epoch: 715 |

MeanAbsoluteError: 0.0889981612563133 | Loss: 0.0135131765469365 | Epoch: 716 | MeanAbsoluteE

MeanAbsoluteError: 0.0864268839359283 | Loss: 0.0125070090455119 | Epoch: 718 |

MeanAbsoluteError: 0.0849650055170059 | Loss: 0.0120259710117170 | Epoch: 719 | MeanAbsoluteE

MeanAbsoluteError: 0.0917269736528397 | Loss: 0.0137507068733127 | Epoch: 721 |

MeanAbsoluteError: 0.0925088301301003 | Loss: 0.0138548627989803 | Epoch: 722 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.0906502306461334 | Loss: 0.0135431011684129 | Epoch: 724 |

MeanAbsoluteError: 0.0912740007042885 | Loss: 0.0142512933164229 | Epoch: 725 | MeanAbsoluteE

MeanAbsoluteError: 0.0925178974866867 | Loss: 0.0142115930517321 | Epoch: 727 |

MeanAbsoluteError: 0.0906669571995735 | Loss: 0.0142422281684897 | Epoch: 728 | MeanAbsoluteE

MeanAbsoluteError: 0.0927450954914093 | Loss: 0.0144200655653064 | Epoch: 730 |

MeanAbsoluteError: 0.0886863693594933 | Loss: 0.0138317218593632 | Epoch: 731 | MeanAbsoluteE

MeanAbsoluteError: 0.0907589122653008 | Loss: 0.0141121127679556 | Epoch: 733 |

MeanAbsoluteError: 0.0871599838137627 | Loss: 0.0132478065496737 | Epoch: 734 | MeanAbsoluteE

MeanAbsoluteError: 0.0829025357961655 | Loss: 0.0122426508738621 | Epoch: 736 |

MeanAbsoluteError: 0.0877145305275917 | Loss: 0.0133188602299197 | Epoch: 737 | MeanAbsoluteE

MeanAbsoluteError: 0.0845664516091347 | Loss: 0.0120545145230183 | Epoch: 739 |

MeanAbsoluteError: 0.0970773026347160 | Loss: 0.0151720033465729 | Epoch: 740 | MeanAbsoluteE

MeanAbsoluteError: 0.0893263593316078 | Loss: 0.0141855918039073 | Epoch: 742 |

MeanAbsoluteError: 0.0978976413607597 | Loss: 0.0167581577059900 | Epoch: 743 | MeanAbsoluteE

MeanAbsoluteError: 0.0926013514399529 | Loss: 0.0151801425500889 | Epoch: 745 |

MeanAbsoluteError: 0.0899734497070312 | Loss: 0.0131786525214557 | Epoch: 746 | MeanAbsoluteE

MeanAbsoluteError: 0.0856569111347198 | Loss: 0.0120810162642677 | Epoch: 748 |

MeanAbsoluteError: 0.0952286943793297 | Loss: 0.0153566859836671 | Epoch: 749 | MeanAbsoluteE

MeanAbsoluteError: 0.0878639742732048 | Loss: 0.0136573054444549 | Epoch: 751 |
```

```
MeanAbsoluteError: 0.0869896262884140 | Loss: 0.0137936002490460 | Epoch: 752 | MeanAbsoluteE

MeanAbsoluteError: 0.0857563689351082 | Loss: 0.0128866208631371 | Epoch: 754 |

MeanAbsoluteError: 0.0900902822613716 | Loss: 0.0137869871610989 | Epoch: 755 | MeanAbsoluteE

MeanAbsoluteError: 0.0848853960633278 | Loss: 0.0121409312021084 | Epoch: 757 |

MeanAbsoluteError: 0.0911357328295708 | Loss: 0.0135928208857755 | Epoch: 758 | MeanAbsoluteE

MeanAbsoluteError: 0.0942618027329445 | Loss: 0.0149122432687242 | Epoch: 760 |

MeanAbsoluteError: 0.0883929282426834 | Loss: 0.0128692606805953 | Epoch: 761 | MeanAbsoluteE

MeanAbsoluteError: 0.0886494666337967 | Loss: 0.0142555460269432 | Epoch: 763 |

MeanAbsoluteError: 0.0915718376636505 | Loss: 0.0145314316730946 | Epoch: 764 | MeanAbsoluteE

MeanAbsoluteError: 0.0893748849630356 | Loss: 0.0134678121875913 | Epoch: 766 |

MeanAbsoluteError: 0.0888227522373199 | Loss: 0.0141681184797199 | Epoch: 767 | MeanAbsoluteE

MeanAbsoluteError: 0.0884083062410355 | Loss: 0.0131590022883999 | Epoch: 769 |

MeanAbsoluteError: 0.0906369686126709 | Loss: 0.0135114987640069 | Epoch: 770 | MeanAbsoluteE

MeanAbsoluteError: 0.0845366641879082 | Loss: 0.0124210573475769 | Epoch: 772 |

MeanAbsoluteError: 0.0926522836089134 | Loss: 0.0144905764923654 | Epoch: 773 | MeanAbsoluteE

MeanAbsoluteError: 0.0874975398182869 | Loss: 0.0138024844552751 | Epoch: 775 |

MeanAbsoluteError: 0.0837405622005463 | Loss: 0.0118689921294087 | Epoch: 776 | MeanAbsoluteE

MeanAbsoluteError: 0.0881681665778160 | Loss: 0.0135657825084248 | Epoch: 778 |

MeanAbsoluteError: 0.0843274965882301 | Loss: 0.0121743180356862 | Epoch: 779 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.0824416577816010 | Loss: 0.0119652940010322 | Epoch: 781 |

MeanAbsoluteError: 0.0863908529281616 | Loss: 0.0132538991586383 | Epoch: 782 | MeanAbsolutel

MeanAbsoluteError: 0.0863013863563538 | Loss: 0.0121469993728776 | Epoch: 784 |

MeanAbsoluteError: 0.0837292373180389 | Loss: 0.0121090052489793 | Epoch: 785 | MeanAbsolutel

MeanAbsoluteError: 0.0827151909470558 | Loss: 0.0119705001153731 | Epoch: 787 |

MeanAbsoluteError: 0.0859255045652390 | Loss: 0.0120229470438790 | Epoch: 788 | MeanAbsolutel

MeanAbsoluteError: 0.0836640670895576 | Loss: 0.0119103719897491 | Epoch: 790 |

MeanAbsoluteError: 0.0850742831826210 | Loss: 0.0132595096883233 | Epoch: 791 | MeanAbsolutel

MeanAbsoluteError: 0.0847994238138199 | Loss: 0.0128509620719463 | Epoch: 793 |

MeanAbsoluteError: 0.0860765278339386 | Loss: 0.0130342804547884 | Epoch: 794 | MeanAbsolutel

MeanAbsoluteError: 0.0825424417853355 | Loss: 0.0119415889284149 | Epoch: 796 |

MeanAbsoluteError: 0.0870050936937332 | Loss: 0.0133550742031366 | Epoch: 797 | MeanAbsolutel

MeanAbsoluteError: 0.0872126296162605 | Loss: 0.0146164064191392 | Epoch: 799 |

MeanAbsoluteError: 0.0880660340189934 | Loss: 0.0134773695795411 | Epoch: 800 | MeanAbsolutel

MeanAbsoluteError: 0.0864510759711266 | Loss: 0.0124022850206044 | Epoch: 802 |

MeanAbsoluteError: 0.0835570916533470 | Loss: 0.0122279131700634 | Epoch: 803 | MeanAbsolutel

MeanAbsoluteError: 0.0811419039964676 | Loss: 0.0114942902098937 | Epoch: 805 |

MeanAbsoluteError: 0.0858493819832802 | Loss: 0.0124588700999690 | Epoch: 806 | MeanAbsolutel

MeanAbsoluteError: 0.0861593410372734 | Loss: 0.0130614097197637 | Epoch: 808 |
```

```
MeanAbsoluteError: 0.0802581608295441 | Loss: 0.0121338178647481 | Epoch: 809 | MeanAbsoluteE

MeanAbsoluteError: 0.0870948955416679 | Loss: 0.0141608328840812 | Epoch: 811 |

MeanAbsoluteError: 0.0813717842102051 | Loss: 0.0106794115207352 | Epoch: 812 | MeanAbsoluteE

MeanAbsoluteError: 0.0842069908976555 | Loss: 0.0120214894860207 | Epoch: 814 |

MeanAbsoluteError: 0.0844162255525589 | Loss: 0.0129236855429948 | Epoch: 815 | MeanAbsoluteE

MeanAbsoluteError: 0.0882782191038132 | Loss: 0.0139897393455127 | Epoch: 817 |

MeanAbsoluteError: 0.0847960188984871 | Loss: 0.0125936034666423 | Epoch: 818 | MeanAbsoluteE

MeanAbsoluteError: 0.0865228548645973 | Loss: 0.0129216267424636 | Epoch: 820 |

MeanAbsoluteError: 0.0908525511622429 | Loss: 0.0144469811899520 | Epoch: 821 | MeanAbsoluteE

MeanAbsoluteError: 0.0838093385100365 | Loss: 0.0126427187522010 | Epoch: 823 |

MeanAbsoluteError: 0.0876276940107346 | Loss: 0.0128317003705888 | Epoch: 824 | MeanAbsoluteE

MeanAbsoluteError: 0.0837254300713539 | Loss: 0.0116518675555926 | Epoch: 826 |

MeanAbsoluteError: 0.0907218828797340 | Loss: 0.0143867454093318 | Epoch: 827 | MeanAbsoluteE

MeanAbsoluteError: 0.0818912908434868 | Loss: 0.0114546013905783 | Epoch: 829 |

MeanAbsoluteError: 0.0814923346042633 | Loss: 0.0117779902179124 | Epoch: 830 | MeanAbsoluteE

MeanAbsoluteError: 0.0837153270840645 | Loss: 0.0120156731203557 | Epoch: 832 |

MeanAbsoluteError: 0.0856942459940910 | Loss: 0.0130324495107440 | Epoch: 833 | MeanAbsoluteE

MeanAbsoluteError: 0.0795649215579033 | Loss: 0.0111408531886991 | Epoch: 835 |

MeanAbsoluteError: 0.0865771174430847 | Loss: 0.0134704502473278 | Epoch: 836 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.0881215631961823 | Loss: 0.0133381166981417 | Epoch: 838 |

MeanAbsoluteError: 0.0809364616870880 | Loss: 0.0108184090172290 | Epoch: 839 | MeanAbsolutel

MeanAbsoluteError: 0.0868527516722679 | Loss: 0.0129571315632590 | Epoch: 841 |

MeanAbsoluteError: 0.0854811891913414 | Loss: 0.0141361958110307 | Epoch: 842 | MeanAbsolutel

MeanAbsoluteError: 0.0867347866296768 | Loss: 0.0130909374239127 | Epoch: 844 |

MeanAbsoluteError: 0.0818386599421501 | Loss: 0.0119324701667362 | Epoch: 845 | MeanAbsolutel

MeanAbsoluteError: 0.0792172104120255 | Loss: 0.0107778623301965 | Epoch: 847 |

MeanAbsoluteError: 0.0829781740903854 | Loss: 0.0120624217230215 | Epoch: 848 | MeanAbsolutel

MeanAbsoluteError: 0.0837435126304626 | Loss: 0.0130439610876298 | Epoch: 850 |

MeanAbsoluteError: 0.0777818411588669 | Loss: 0.0098406087394202 | Epoch: 851 | MeanAbsolutel

MeanAbsoluteError: 0.0868685021996498 | Loss: 0.0143707775490566 | Epoch: 853 |

MeanAbsoluteError: 0.0877344012260437 | Loss: 0.0140737598385992 | Epoch: 854 | MeanAbsolutel

MeanAbsoluteError: 0.0832599923014641 | Loss: 0.0120852595238224 | Epoch: 856 |

MeanAbsoluteError: 0.0854643955826759 | Loss: 0.0125624769905698 | Epoch: 857 | MeanAbsolutel

MeanAbsoluteError: 0.0866518318653107 | Loss: 0.0126182180148802 | Epoch: 859 |

MeanAbsoluteError: 0.0843079388141632 | Loss: 0.0120376646205841 | Epoch: 860 | MeanAbsolutel

MeanAbsoluteError: 0.0803577825427055 | Loss: 0.0110632520802513 | Epoch: 862 |

MeanAbsoluteError: 0.0879406630992889 | Loss: 0.0129482736233816 | Epoch: 863 | MeanAbsolutel

MeanAbsoluteError: 0.0869652703404427 | Loss: 0.0129983409093499 | Epoch: 865 |
```

```
MeanAbsoluteError: 0.0793773978948593 | Loss: 0.0108778996478456 | Epoch: 866 | MeanAbsoluteE

MeanAbsoluteError: 0.0727204158902168 | Loss: 0.0091950179879435 | Epoch: 868 |

MeanAbsoluteError: 0.0852394402027130 | Loss: 0.0122983297423343 | Epoch: 869 | MeanAbsoluteE

MeanAbsoluteError: 0.0799776613712311 | Loss: 0.0118252016472828 | Epoch: 871 |

MeanAbsoluteError: 0.0810522809624672 | Loss: 0.0108049600334804 | Epoch: 872 | MeanAbsoluteE

MeanAbsoluteError: 0.0834995657205582 | Loss: 0.0125072591833790 | Epoch: 874 |

MeanAbsoluteError: 0.0789700448513031 | Loss: 0.0112399914978596 | Epoch: 875 | MeanAbsoluteE

MeanAbsoluteError: 0.0839084461331367 | Loss: 0.0122934163055227 | Epoch: 877 |

MeanAbsoluteError: 0.0884987860918045 | Loss: 0.0136814159659737 | Epoch: 878 | MeanAbsoluteE

MeanAbsoluteError: 0.0838527455925941 | Loss: 0.0129079650566564 | Epoch: 880 |

MeanAbsoluteError: 0.0779891982674599 | Loss: 0.0111646611295388 | Epoch: 881 | MeanAbsoluteE

MeanAbsoluteError: 0.0846373289823532 | Loss: 0.0127930974611688 | Epoch: 883 |

MeanAbsoluteError: 0.0761989131569862 | Loss: 0.0100255685634329 | Epoch: 884 | MeanAbsoluteE

MeanAbsoluteError: 0.0887425020337105 | Loss: 0.0137050725410033 | Epoch: 886 |

MeanAbsoluteError: 0.0844006612896919 | Loss: 0.0125558544683736 | Epoch: 887 | MeanAbsoluteE

MeanAbsoluteError: 0.0815278515219688 | Loss: 0.0108900653424401 | Epoch: 889 |

MeanAbsoluteError: 0.0801741555333138 | Loss: 0.0112773405753978 | Epoch: 890 | MeanAbsoluteE

MeanAbsoluteError: 0.0804894566535950 | Loss: 0.0109850597259356 | Epoch: 892 |

MeanAbsoluteError: 0.0847748368978500 | Loss: 0.0128239777176835 | Epoch: 893 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.0747078284621239 | Loss: 0.0102748880828100 | Epoch: 895 |

MeanAbsoluteError: 0.0768106952309608 | Loss: 0.0106994809596654 | Epoch: 896 | MeanAbsolutel

MeanAbsoluteError: 0.0842396914958954 | Loss: 0.0119551145335754 | Epoch: 898 |

MeanAbsoluteError: 0.0858636721968651 | Loss: 0.0124528025139686 | Epoch: 899 | MeanAbsolutel

MeanAbsoluteError: 0.0816988572478294 | Loss: 0.0122027543124568 | Epoch: 901 |

MeanAbsoluteError: 0.0855543464422226 | Loss: 0.0131323056053952 | Epoch: 902 | MeanAbsolutel

MeanAbsoluteError: 0.0812320113182068 | Loss: 0.0117353786758031 | Epoch: 904 |

MeanAbsoluteError: 0.0815234631299973 | Loss: 0.0117177357265609 | Epoch: 905 | MeanAbsolutel

MeanAbsoluteError: 0.0821851491928101 | Loss: 0.0114739820748703 | Epoch: 907 |

MeanAbsoluteError: 0.0806851387023926 | Loss: 0.0112064469993735 | Epoch: 908 | MeanAbsolutel

MeanAbsoluteError: 0.0845059305429459 | Loss: 0.0131811214650709 | Epoch: 910 |

MeanAbsoluteError: 0.0840513110160828 | Loss: 0.0126629632166199 | Epoch: 911 | MeanAbsolutel

MeanAbsoluteError: 0.0818073675036430 | Loss: 0.0121243475854862 | Epoch: 913 |

MeanAbsoluteError: 0.0823264941573143 | Loss: 0.0126068318631345 | Epoch: 914 | MeanAbsolutel

MeanAbsoluteError: 0.0864855125546455 | Loss: 0.0121105131108519 | Epoch: 916 |

MeanAbsoluteError: 0.0806026682257652 | Loss: 0.0108642235760817 | Epoch: 917 | MeanAbsolutel

MeanAbsoluteError: 0.0831850916147232 | Loss: 0.0118972804475925 | Epoch: 919 |

MeanAbsoluteError: 0.0859341472387314 | Loss: 0.0123188406359138 | Epoch: 920 | MeanAbsolutel

MeanAbsoluteError: 0.0769573450088501 | Loss: 0.0110629977945549 | Epoch: 922 |
```

```
MeanAbsoluteError: 0.0817093774676323 | Loss: 0.0115376829734062 | Epoch: 923 | MeanAbsoluteE

MeanAbsoluteError: 0.0833878666162491 | Loss: 0.0120054585821648 | Epoch: 925 |

MeanAbsoluteError: 0.0830642282962799 | Loss: 0.0123982455462101 | Epoch: 926 | MeanAbsoluteE

MeanAbsoluteError: 0.0824942737817764 | Loss: 0.0110520471071747 | Epoch: 928 |

MeanAbsoluteError: 0.0794909298419952 | Loss: 0.0110053960340883 | Epoch: 929 | MeanAbsoluteE

MeanAbsoluteError: 0.0751781687140465 | Loss: 0.0095843236330984 | Epoch: 931 |

MeanAbsoluteError: 0.0836121961474419 | Loss: 0.0119021516470821 | Epoch: 932 | MeanAbsoluteE

MeanAbsoluteError: 0.0821736007928848 | Loss: 0.0123494721528308 | Epoch: 934 |

MeanAbsoluteError: 0.0776700153946877 | Loss: 0.0106821060330064 | Epoch: 935 | MeanAbsoluteE

MeanAbsoluteError: 0.0787449181079865 | Loss: 0.0110101186536485 | Epoch: 937 |

MeanAbsoluteError: 0.0783307179808617 | Loss: 0.0107948462445347 | Epoch: 938 | MeanAbsoluteE

MeanAbsoluteError: 0.0779344066977501 | Loss: 0.0105653461959446 | Epoch: 940 |

MeanAbsoluteError: 0.0825816839933395 | Loss: 0.0116984023561236 | Epoch: 941 | MeanAbsoluteE

MeanAbsoluteError: 0.0837830454111099 | Loss: 0.0126777350001006 | Epoch: 943 |

MeanAbsoluteError: 0.0850668326020241 | Loss: 0.0125878079836411 | Epoch: 944 | MeanAbsoluteE

MeanAbsoluteError: 0.0759390369057655 | Loss: 0.0105397758814130 | Epoch: 946 |

MeanAbsoluteError: 0.0828461945056915 | Loss: 0.0121800232877528 | Epoch: 947 | MeanAbsoluteE

MeanAbsoluteError: 0.0790953859686852 | Loss: 0.0112134233638062 | Epoch: 949 |

MeanAbsoluteError: 0.0718504562973976 | Loss: 0.0090638517100858 | Epoch: 950 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.0713303685188293 | Loss: 0.0092097786820129 | Epoch: 952 |
MeanAbsoluteError: 0.0758384764194489 | Loss: 0.0105704335976528 | Epoch: 953 | MeanAbsolutel
MeanAbsoluteError: 0.0764975696802139 | Loss: 0.0110457399899557 | Epoch: 955 |
MeanAbsoluteError: 0.0776337683200836 | Loss: 0.0100981653131263 | Epoch: 956 | MeanAbsolutel
MeanAbsoluteError: 0.0788982808589935 | Loss: 0.0109099217089897 | Epoch: 958 |
MeanAbsoluteError: 0.0820529311895370 | Loss: 0.0123588436651820 | Epoch: 959 | MeanAbsolutel
MeanAbsoluteError: 0.0807428881525993 | Loss: 0.0116530753388967 | Epoch: 961 |
MeanAbsoluteError: 0.0810418501496315 | Loss: 0.0115379188655061 | Epoch: 962 | MeanAbsolutel
MeanAbsoluteError: 0.0808367505669594 | Loss: 0.0114152176126784 | Epoch: 964 |
MeanAbsoluteError: 0.0748266875743866 | Loss: 0.0102052634650803 | Epoch: 965 | MeanAbsolutel
MeanAbsoluteError: 0.0813763216137886 | Loss: 0.0112484617283189 | Epoch: 967 |
MeanAbsoluteError: 0.0752978622913361 | Loss: 0.0109890507715803 | Epoch: 968 | MeanAbsolutel
MeanAbsoluteError: 0.0761685743927956 | Loss: 0.0098923540637770 | Epoch: 970 |
MeanAbsoluteError: 0.0784949585795403 | Loss: 0.0114520865059664 | Epoch: 971 | MeanAbsolutel
MeanAbsoluteError: 0.0732161402702332 | Loss: 0.0101727204033614 | Epoch: 973 |
MeanAbsoluteError: 0.0834525227546692 | Loss: 0.0113418011025836 | Epoch: 974 | MeanAbsolutel
MeanAbsoluteError: 0.0828372910618782 | Loss: 0.0116602848018617 | Epoch: 976 |
MeanAbsoluteError: 0.0750916898250580 | Loss: 0.0103996038327750 | Epoch: 977 | MeanAbsolutel
MeanAbsoluteError: 0.0755204260349274 | Loss: 0.0096732087005269 | Epoch: 979 |
```

```
MeanAbsoluteError: 0.0808703079819679 | Loss: 0.0109301538461417 | Epoch: 980 | MeanAbsolutel

MeanAbsoluteError: 0.0740360170602798 | Loss: 0.0105949369321024 | Epoch: 982 |

MeanAbsoluteError: 0.0782309845089912 | Loss: 0.0107162922217200 | Epoch: 983 | MeanAbsolutel

MeanAbsoluteError: 0.0796489641070366 | Loss: 0.0108856909895925 | Epoch: 985 |

MeanAbsoluteError: 0.0754445046186447 | Loss: 0.0101576442988153 | Epoch: 986 | MeanAbsolutel

MeanAbsoluteError: 0.0749045684933662 | Loss: 0.0099742748832796 | Epoch: 988 |

MeanAbsoluteError: 0.0750370174646378 | Loss: 0.0097567541729222 | Epoch: 989 | MeanAbsolutel

MeanAbsoluteError: 0.0798285901546478 | Loss: 0.0110966821341087 | Epoch: 991 |

MeanAbsoluteError: 0.0784056931734085 | Loss: 0.0100535742992846 | Epoch: 992 | MeanAbsolutel

MeanAbsoluteError: 0.0803987458348274 | Loss: 0.0113317678580885 | Epoch: 994 |

MeanAbsoluteError: 0.0752013847231865 | Loss: 0.0098552726883463 | Epoch: 995 | MeanAbsolutel

MeanAbsoluteError: 0.0815056562423706 | Loss: 0.0120711852494181 | Epoch: 997 |

MeanAbsoluteError: 0.0818167477846146 | Loss: 0.0121984705777140 | Epoch: 998 | MeanAbsolutel

MeanAbsoluteError: 0.0783293992280960 | Loss: 0.0113129563942978 | Epoch: 1000 |

MeanAbsoluteError: 0.0756960287690163 | Loss: 0.0094856642690199 | Epoch: 1001 | MeanAbsolute

MeanAbsoluteError: 0.0761614143848419 | Loss: 0.0101154720262275 | Epoch: 1003 |

MeanAbsoluteError: 0.0767387002706528 | Loss: 0.0106298152299981 | Epoch: 1004 | MeanAbsolute

MeanAbsoluteError: 0.0766677260398865 | Loss: 0.0101231759442514 | Epoch: 1006 |

MeanAbsoluteError: 0.0793050751090050 | Loss: 0.0112250647338684 | Epoch: 1007 | MeanAbsolute
```

```
MeanAbsoluteError: 0.0713485553860664 | Loss: 0.0091363125214896 | Epoch: 1009 |

MeanAbsoluteError: 0.0705844908952713 | Loss: 0.0090202878686190 | Epoch: 1010 | MeanAbsolute

MeanAbsoluteError: 0.0769069641828537 | Loss: 0.0103006165401894 | Epoch: 1012 |

MeanAbsoluteError: 0.0699821710586548 | Loss: 0.0091856122973453 | Epoch: 1013 | MeanAbsolute

MeanAbsoluteError: 0.0768254920840263 | Loss: 0.0107477369990859 | Epoch: 1015 |

MeanAbsoluteError: 0.0723410174250603 | Loss: 0.0100517477534595 | Epoch: 1016 | MeanAbsolute

MeanAbsoluteError: 0.0818216726183891 | Loss: 0.0118001714335605 | Epoch: 1018 |

MeanAbsoluteError: 0.0784135237336159 | Loss: 0.0108611970977169 | Epoch: 1019 | MeanAbsolute

MeanAbsoluteError: 0.0780879482626915 | Loss: 0.0109470508112766 | Epoch: 1021 |

MeanAbsoluteError: 0.0704880878329277 | Loss: 0.0087977734877495 | Epoch: 1022 | MeanAbsolute

MeanAbsoluteError: 0.0720255672931671 | Loss: 0.0091468593972119 | Epoch: 1024 |

MeanAbsoluteError: 0.0775656551122665 | Loss: 0.0109609235635920 | Epoch: 1025 | MeanAbsolute

MeanAbsoluteError: 0.0820410847663879 | Loss: 0.0120141109563701 | Epoch: 1027 |

MeanAbsoluteError: 0.0724329203367233 | Loss: 0.0101661229858049 | Epoch: 1028 | MeanAbsolute

MeanAbsoluteError: 0.0810569003224373 | Loss: 0.0108305043519552 | Epoch: 1030 |

MeanAbsoluteError: 0.0741794556379318 | Loss: 0.0099986493321558 | Epoch: 1031 | MeanAbsolute

MeanAbsoluteError: 0.0731076970696449 | Loss: 0.0090496563608758 | Epoch: 1033 |

MeanAbsoluteError: 0.0737868398427963 | Loss: 0.0096263391597919 | Epoch: 1034 | MeanAbsolute

MeanAbsoluteError: 0.0719187781214714 | Loss: 0.0093971768439466 | Epoch: 1036 |
```

```
MeanAbsoluteError: 0.0783430188894272 | Loss: 0.0112729077730910 | Epoch: 1037 | MeanAbsolute

MeanAbsoluteError: 0.0731714591383934 | Loss: 0.0098591697473845 | Epoch: 1039 |

MeanAbsoluteError: 0.0755417793989182 | Loss: 0.0103592779876635 | Epoch: 1040 | MeanAbsolute

MeanAbsoluteError: 0.0726536735892296 | Loss: 0.0094302292333062 | Epoch: 1042 |

MeanAbsoluteError: 0.0722122862935066 | Loss: 0.0095080077463354 | Epoch: 1043 | MeanAbsolute

MeanAbsoluteError: 0.0781117379665375 | Loss: 0.0106983402077458 | Epoch: 1045 |

MeanAbsoluteError: 0.0735459029674530 | Loss: 0.0107289597397660 | Epoch: 1046 | MeanAbsolute

MeanAbsoluteError: 0.0736163705587387 | Loss: 0.0095382951039574 | Epoch: 1048 |

MeanAbsoluteError: 0.0745738819241524 | Loss: 0.0097680380151238 | Epoch: 1049 | MeanAbsolute

MeanAbsoluteError: 0.0784629806876183 | Loss: 0.0115093758331750 | Epoch: 1051 |

MeanAbsoluteError: 0.0708526596426964 | Loss: 0.0089048978778980 | Epoch: 1052 | MeanAbsolute

MeanAbsoluteError: 0.0762671753764153 | Loss: 0.0104403722548886 | Epoch: 1054 |

MeanAbsoluteError: 0.0715388879179955 | Loss: 0.0098808641400986 | Epoch: 1055 | MeanAbsolute

MeanAbsoluteError: 0.0724579021334648 | Loss: 0.0102651088655208 | Epoch: 1057 |

MeanAbsoluteError: 0.0732223764061928 | Loss: 0.0094526149104422 | Epoch: 1058 | MeanAbsolute

MeanAbsoluteError: 0.0764526650309563 | Loss: 0.0098578167982851 | Epoch: 1060 |

MeanAbsoluteError: 0.0705185234546661 | Loss: 0.0091684215193557 | Epoch: 1061 | MeanAbsolute

MeanAbsoluteError: 0.0742731764912605 | Loss: 0.0106905071593064 | Epoch: 1063 |

MeanAbsoluteError: 0.0752537027001381 | Loss: 0.0094573750439918 | Epoch: 1064 | MeanAbsolute
```

```
MeanAbsoluteError: 0.0752584338188171 | Loss: 0.0105336007947456 | Epoch: 1066 |
MeanAbsoluteError: 0.0727235525846481 | Loss: 0.0101038878907760 | Epoch: 1067 | MeanAbsolute
MeanAbsoluteError: 0.0736334547400475 | Loss: 0.0097346583839559 | Epoch: 1069 |
MeanAbsoluteError: 0.0750995352864265 | Loss: 0.0101968461093141 | Epoch: 1070 | MeanAbsolute
MeanAbsoluteError: 0.0743859335780144 | Loss: 0.0100097836365270 | Epoch: 1072 |
MeanAbsoluteError: 0.0720578059554100 | Loss: 0.0093653451314943 | Epoch: 1073 | MeanAbsolute
MeanAbsoluteError: 0.0723907500505447 | Loss: 0.0094384221381794 | Epoch: 1075 |
MeanAbsoluteError: 0.0701497346162796 | Loss: 0.0091049439610409 | Epoch: 1076 | MeanAbsolute
MeanAbsoluteError: 0.0713358670473099 | Loss: 0.0095145386230729 | Epoch: 1078 |
MeanAbsoluteError: 0.0744606554508209 | Loss: 0.0097779069569515 | Epoch: 1079 | MeanAbsolute
MeanAbsoluteError: 0.0745055899024010 | Loss: 0.0096419711591928 | Epoch: 1081 |
MeanAbsoluteError: 0.0750945135951042 | Loss: 0.0115239816393538 | Epoch: 1082 | MeanAbsolute
MeanAbsoluteError: 0.0726505741477013 | Loss: 0.0092043116474573 | Epoch: 1084 |
MeanAbsoluteError: 0.0738136470317841 | Loss: 0.0101096401996953 | Epoch: 1085 | MeanAbsolute
MeanAbsoluteError: 0.0714196860790253 | Loss: 0.0096742579741597 | Epoch: 1087 |
MeanAbsoluteError: 0.0736794769763947 | Loss: 0.0096946267451373 | Epoch: 1088 | MeanAbsolute
MeanAbsoluteError: 0.0717932730913162 | Loss: 0.0096871913681753 | Epoch: 1090 |
MeanAbsoluteError: 0.0742963477969170 | Loss: 0.0094376059675900 | Epoch: 1091 | MeanAbsolute
MeanAbsoluteError: 0.0719407498836517 | Loss: 0.0088017874735063 | Epoch: 1093 |
```

```
MeanAbsoluteError: 0.0720498561859131 | Loss: 0.0096926858180025 | Epoch: 1094 | MeanAbsolute

MeanAbsoluteError: 0.0740646943449974 | Loss: 0.0101780233704509 | Epoch: 1096 |

MeanAbsoluteError: 0.0668474361300468 | Loss: 0.0084361395483211 | Epoch: 1097 | MeanAbsolute

MeanAbsoluteError: 0.0756465792655945 | Loss: 0.0104626348147576 | Epoch: 1099 |

MeanAbsoluteError: 0.0759581327438354 | Loss: 0.0101107480868814 | Epoch: 1100 | MeanAbsolute

MeanAbsoluteError: 0.0713522955775261 | Loss: 0.0089964508296544 | Epoch: 1102 |

MeanAbsoluteError: 0.0748826190829277 | Loss: 0.0100760073738881 | Epoch: 1103 | MeanAbsolute

MeanAbsoluteError: 0.0765188708901405 | Loss: 0.0108012736362980 | Epoch: 1105 |

MeanAbsoluteError: 0.0692346021533012 | Loss: 0.0088205982657958 | Epoch: 1106 | MeanAbsolute

MeanAbsoluteError: 0.0754934027791023 | Loss: 0.0100899146969702 | Epoch: 1108 |

MeanAbsoluteError: 0.0776676535606384 | Loss: 0.0104353230330404 | Epoch: 1109 | MeanAbsolute

MeanAbsoluteError: 0.0651612505316734 | Loss: 0.0080974475529365 | Epoch: 1111 |

MeanAbsoluteError: 0.0727598071098328 | Loss: 0.0093571880667029 | Epoch: 1112 | MeanAbsolute

MeanAbsoluteError: 0.0717332363128662 | Loss: 0.0099709212735373 | Epoch: 1114 |

MeanAbsoluteError: 0.0724366009235382 | Loss: 0.0088772476846255 | Epoch: 1115 | MeanAbsolute

MeanAbsoluteError: 0.0686012357473373 | Loss: 0.0082778730559706 | Epoch: 1117 |

MeanAbsoluteError: 0.0796457827091217 | Loss: 0.0112244006519298 | Epoch: 1118 | MeanAbsolute

MeanAbsoluteError: 0.0730471611022949 | Loss: 0.0095126135364020 | Epoch: 1120 |

MeanAbsoluteError: 0.0725330337882042 | Loss: 0.0094914949694006 | Epoch: 1121 | MeanAbsolute
```

```
MeanAbsoluteError: 0.0765260234475136 | Loss: 0.0106466337969080 | Epoch: 1123 |

MeanAbsoluteError: 0.0737615823745728 | Loss: 0.0099871392685721 | Epoch: 1124 | MeanAbsolute

MeanAbsoluteError: 0.0706823542714119 | Loss: 0.0095593848242182 | Epoch: 1126 |

MeanAbsoluteError: 0.0745585858821869 | Loss: 0.0093951427221024 | Epoch: 1127 | MeanAbsolute

MeanAbsoluteError: 0.0701927244663239 | Loss: 0.0092704805639854 | Epoch: 1129 |

MeanAbsoluteError: 0.0687992647290230 | Loss: 0.0088551788672097 | Epoch: 1130 | MeanAbsolute

MeanAbsoluteError: 0.0698541998863220 | Loss: 0.0086771417512258 | Epoch: 1132 |

MeanAbsoluteError: 0.0705801323056221 | Loss: 0.0089557803918433 | Epoch: 1133 | MeanAbsolute

MeanAbsoluteError: 0.0660745799541473 | Loss: 0.0078033145223298 | Epoch: 1135 |

MeanAbsoluteError: 0.0697684213519096 | Loss: 0.0093432867652640 | Epoch: 1136 | MeanAbsolute

MeanAbsoluteError: 0.0707690641283989 | Loss: 0.0091020797718704 | Epoch: 1138 |

MeanAbsoluteError: 0.0721527487039566 | Loss: 0.0090801251660256 | Epoch: 1139 | MeanAbsolute

MeanAbsoluteError: 0.0726815909147263 | Loss: 0.0103949950547152 | Epoch: 1141 |

MeanAbsoluteError: 0.0686275213956833 | Loss: 0.0078791715283660 | Epoch: 1142 | MeanAbsolute

MeanAbsoluteError: 0.0686838999390602 | Loss: 0.0083464341980471 | Epoch: 1144 |

MeanAbsoluteError: 0.0722418501973152 | Loss: 0.0096947181629074 | Epoch: 1145 | MeanAbsolute

MeanAbsoluteError: 0.0718059688806534 | Loss: 0.0087835784534885 | Epoch: 1147 |

MeanAbsoluteError: 0.0718396082520485 | Loss: 0.0098031430166642 | Epoch: 1148 | MeanAbsolute

MeanAbsoluteError: 0.0689787715673447 | Loss: 0.0086593381095978 | Epoch: 1150 |
```

```
MeanAbsoluteError: 0.0680249556899071 | Loss: 0.0084708573133200 | Epoch: 1151 | MeanAbsolute

MeanAbsoluteError: 0.0763657540082932 | Loss: 0.0098982771009226 | Epoch: 1153 |

MeanAbsoluteError: 0.0718902871012688 | Loss: 0.0089265587373181 | Epoch: 1154 | MeanAbsolute

MeanAbsoluteError: 0.0669967979192734 | Loss: 0.0080683415926372 | Epoch: 1156 |

MeanAbsoluteError: 0.0685659945011139 | Loss: 0.0090044095931201 | Epoch: 1157 | MeanAbsolute

MeanAbsoluteError: 0.0736594498157501 | Loss: 0.0097183153880299 | Epoch: 1159 |

MeanAbsoluteError: 0.0752081498503685 | Loss: 0.0097305369102590 | Epoch: 1160 | MeanAbsolute

MeanAbsoluteError: 0.0709177702665329 | Loss: 0.0099466601758938 | Epoch: 1162 |

MeanAbsoluteError: 0.0702000334858894 | Loss: 0.0094173097604895 | Epoch: 1163 | MeanAbsolute

MeanAbsoluteError: 0.0646093636751175 | Loss: 0.0073754562852567 | Epoch: 1165 |

MeanAbsoluteError: 0.0718458518385887 | Loss: 0.0101621998156558 | Epoch: 1166 | MeanAbsolute

MeanAbsoluteError: 0.0709827765822411 | Loss: 0.0090457766425000 | Epoch: 1168 |

MeanAbsoluteError: 0.0697078108787537 | Loss: 0.0085737795314587 | Epoch: 1169 | MeanAbsolute

MeanAbsoluteError: 0.0698874890804291 | Loss: 0.0090111857553620 | Epoch: 1171 |

MeanAbsoluteError: 0.0706130936741829 | Loss: 0.0098039477243825 | Epoch: 1172 | MeanAbsolute

MeanAbsoluteError: 0.0680111721158028 | Loss: 0.0086102833707264 | Epoch: 1174 |

MeanAbsoluteError: 0.0712836608290672 | Loss: 0.0090126027286169 | Epoch: 1175 | MeanAbsolute

MeanAbsoluteError: 0.0697515085339546 | Loss: 0.0085084578614139 | Epoch: 1177 |

MeanAbsoluteError: 0.0739419385790825 | Loss: 0.0104043591892696 | Epoch: 1178 | MeanAbsolute
```

```
MeanAbsoluteError: 0.0701684802770615 | Loss: 0.0086668234991339 | Epoch: 1180 |
MeanAbsoluteError: 0.0662872046232224 | Loss: 0.0079811214613195 | Epoch: 1181 | MeanAbsolute
MeanAbsoluteError: 0.0709202140569687 | Loss: 0.0089032758617153 | Epoch: 1183 |
MeanAbsoluteError: 0.0736245438456535 | Loss: 0.0097737407205295 | Epoch: 1184 | MeanAbsolute
MeanAbsoluteError: 0.0697964131832123 | Loss: 0.0088830038035151 | Epoch: 1186 |
MeanAbsoluteError: 0.0687001422047615 | Loss: 0.0081080730247049 | Epoch: 1187 | MeanAbsolute
MeanAbsoluteError: 0.0732466280460358 | Loss: 0.0110408794836500 | Epoch: 1189 |
MeanAbsoluteError: 0.0676024556159973 | Loss: 0.0080906260354732 | Epoch: 1190 | MeanAbsolute
MeanAbsoluteError: 0.0718645006418228 | Loss: 0.0098977418978272 | Epoch: 1192 |
MeanAbsoluteError: 0.0739581137895584 | Loss: 0.0099091313641960 | Epoch: 1193 | MeanAbsolute
MeanAbsoluteError: 0.0712649971246719 | Loss: 0.0096088448665250 | Epoch: 1195 |
MeanAbsoluteError: 0.0680521130561829 | Loss: 0.0081771077490945 | Epoch: 1196 | MeanAbsolute
MeanAbsoluteError: 0.0684056654572487 | Loss: 0.0079836140158295 | Epoch: 1198 |
MeanAbsoluteError: 0.0693398118019104 | Loss: 0.0087397836947639 | Epoch: 1199 | MeanAbsolute
MeanAbsoluteError: 0.0679291263222694 | Loss: 0.0080958792559977 | Epoch: 1201 |
MeanAbsoluteError: 0.0675697475671768 | Loss: 0.0087066634076958 | Epoch: 1202 | MeanAbsolute
MeanAbsoluteError: 0.0676503702998161 | Loss: 0.0083344452205953 | Epoch: 1204 |
MeanAbsoluteError: 0.0713101401925087 | Loss: 0.0083928035043452 | Epoch: 1205 | MeanAbsolute
MeanAbsoluteError: 0.0668883770704269 | Loss: 0.0079435738545726 | Epoch: 1207 |
```

```
MeanAbsoluteError: 0.0722930952906609 | Loss: 0.0090595198233981 | Epoch: 1208 | MeanAbsolute

MeanAbsoluteError: 0.0659274980425835 | Loss: 0.0079206271164670 | Epoch: 1210 |

MeanAbsoluteError: 0.0711612552404404 | Loss: 0.0093496699937047 | Epoch: 1211 | MeanAbsolute

MeanAbsoluteError: 0.0701391845941544 | Loss: 0.0093934695026352 | Epoch: 1213 |

MeanAbsoluteError: 0.0680515095591545 | Loss: 0.0087290170410779 | Epoch: 1214 | MeanAbsolute

MeanAbsoluteError: 0.0693480074405670 | Loss: 0.0093624994940183 | Epoch: 1216 |

MeanAbsoluteError: 0.0699780359864235 | Loss: 0.0091742996709399 | Epoch: 1217 | MeanAbsolute

MeanAbsoluteError: 0.0698490217328072 | Loss: 0.0092482412282728 | Epoch: 1219 |

MeanAbsoluteError: 0.0642681866884232 | Loss: 0.0073484826516506 | Epoch: 1220 | MeanAbsolute

MeanAbsoluteError: 0.0738641396164894 | Loss: 0.0097432722705707 | Epoch: 1222 |

MeanAbsoluteError: 0.0662506893277168 | Loss: 0.0084515389894417 | Epoch: 1223 | MeanAbsolute

MeanAbsoluteError: 0.0679148435592651 | Loss: 0.0084353626003091 | Epoch: 1225 |

MeanAbsoluteError: 0.0701103284955025 | Loss: 0.0093310862245092 | Epoch: 1226 | MeanAbsolute

MeanAbsoluteError: 0.0693054273724556 | Loss: 0.0090913326680796 | Epoch: 1228 |

MeanAbsoluteError: 0.0652973651885986 | Loss: 0.0082916976060127 | Epoch: 1229 | MeanAbsolute

MeanAbsoluteError: 0.0667857527732849 | Loss: 0.0075740547531556 | Epoch: 1231 |

MeanAbsoluteError: 0.0655782148241997 | Loss: 0.0084831078878688 | Epoch: 1232 | MeanAbsolute

MeanAbsoluteError: 0.0730473473668098 | Loss: 0.0095969933984452 | Epoch: 1234 |

MeanAbsoluteError: 0.0661027133464813 | Loss: 0.0080063621945252 | Epoch: 1235 | MeanAbsolute
```

```
MeanAbsoluteError: 0.0677629560232162 | Loss: 0.0085343337510009 | Epoch: 1237 |
MeanAbsoluteError: 0.0653400346636772 | Loss: 0.0078653017017496 | Epoch: 1238 | MeanAbsolute
MeanAbsoluteError: 0.0667546391487122 | Loss: 0.0086837459862242 | Epoch: 1240 |
MeanAbsoluteError: 0.0678297579288483 | Loss: 0.0083681150305832 | Epoch: 1241 | MeanAbsolute
MeanAbsoluteError: 0.0646193921566010 | Loss: 0.0070690198770293 | Epoch: 1243 |
MeanAbsoluteError: 0.0668182820081711 | Loss: 0.0083145666037065 | Epoch: 1244 | MeanAbsolute
MeanAbsoluteError: 0.0674901753664017 | Loss: 0.0086883052581834 | Epoch: 1246 |
MeanAbsoluteError: 0.0633812770247459 | Loss: 0.0070393207668409 | Epoch: 1247 | MeanAbsolute
MeanAbsoluteError: 0.0670359432697296 | Loss: 0.0094278395879943 | Epoch: 1249 |
MeanAbsoluteError: 0.0656556189060211 | Loss: 0.0082077703897085 | Epoch: 1250 | MeanAbsolute
MeanAbsoluteError: 0.0647459402680397 | Loss: 0.0084532492830840 | Epoch: 1252 |
MeanAbsoluteError: 0.0703307315707207 | Loss: 0.0088216224113179 | Epoch: 1253 | MeanAbsolute
MeanAbsoluteError: 0.0683056563138962 | Loss: 0.0090764825968290 | Epoch: 1255 |
MeanAbsoluteError: 0.0666681528091431 | Loss: 0.0086283355742125 | Epoch: 1256 | MeanAbsolute
MeanAbsoluteError: 0.0643918886780739 | Loss: 0.0078696753475985 | Epoch: 1258 |
MeanAbsoluteError: 0.0665179714560509 | Loss: 0.0085761582433042 | Epoch: 1259 | MeanAbsolute
MeanAbsoluteError: 0.0703458711504936 | Loss: 0.0087383908390626 | Epoch: 1261 |
MeanAbsoluteError: 0.0644440278410912 | Loss: 0.0082479690959250 | Epoch: 1262 | MeanAbsolute
MeanAbsoluteError: 0.0688890516757965 | Loss: 0.0085105541995290 | Epoch: 1264 |
```

```
MeanAbsoluteError: 0.0650466158986092 | Loss: 0.0083929591939038 | Epoch: 1265 | MeanAbsolute

MeanAbsoluteError: 0.0662450492382050 | Loss: 0.0083449949791490 | Epoch: 1267 |

MeanAbsoluteError: 0.0661172196269035 | Loss: 0.0084257720087286 | Epoch: 1268 | MeanAbsolute

MeanAbsoluteError: 0.0697387829422951 | Loss: 0.0093712036786565 | Epoch: 1270 |

MeanAbsoluteError: 0.0644830465316772 | Loss: 0.0081673505102005 | Epoch: 1271 | MeanAbsolute

MeanAbsoluteError: 0.0650059729814529 | Loss: 0.0082024893622596 | Epoch: 1273 |

MeanAbsoluteError: 0.0651058331131935 | Loss: 0.0082260953539632 | Epoch: 1274 | MeanAbsolute

MeanAbsoluteError: 0.0644750893115997 | Loss: 0.0079935343652020 | Epoch: 1276 |

MeanAbsoluteError: 0.0662265568971634 | Loss: 0.0079849014229088 | Epoch: 1277 | MeanAbsolute

MeanAbsoluteError: 0.0647539496421814 | Loss: 0.0083720913629077 | Epoch: 1279 |

MeanAbsoluteError: 0.0675260201096535 | Loss: 0.0082275420509116 | Epoch: 1280 | MeanAbsolute

MeanAbsoluteError: 0.0654538571834564 | Loss: 0.0075902166064528 | Epoch: 1282 |

MeanAbsoluteError: 0.0721336677670479 | Loss: 0.0090969711094901 | Epoch: 1283 | MeanAbsolute

MeanAbsoluteError: 0.0653984993696213 | Loss: 0.0086893718288290 | Epoch: 1285 |

MeanAbsoluteError: 0.0656016319990158 | Loss: 0.0079644805144441 | Epoch: 1286 | MeanAbsolute

MeanAbsoluteError: 0.0685904249548912 | Loss: 0.0089860104821582 | Epoch: 1288 |

MeanAbsoluteError: 0.0689273476600647 | Loss: 0.0081498642442663 | Epoch: 1289 | MeanAbsolute

MeanAbsoluteError: 0.0671072974801064 | Loss: 0.0086835256456592 | Epoch: 1291 |

MeanAbsoluteError: 0.0631592273712158 | Loss: 0.0075809440349015 | Epoch: 1292 | MeanAbsolute
```

```
MeanAbsoluteError: 0.0664058402180672 | Loss: 0.0073231013654731 | Epoch: 1294 |

MeanAbsoluteError: 0.0682762116193771 | Loss: 0.0088728071079368 | Epoch: 1295 | MeanAbsolute

MeanAbsoluteError: 0.0615258961915970 | Loss: 0.0079175249092077 | Epoch: 1297 |

MeanAbsoluteError: 0.0667539685964584 | Loss: 0.0080548802118331 | Epoch: 1298 | MeanAbsolute

MeanAbsoluteError: 0.0634960979223251 | Loss: 0.0073823048852015 | Epoch: 1300 |

MeanAbsoluteError: 0.0693003237247467 | Loss: 0.0087899086087418 | Epoch: 1301 | MeanAbsolute

MeanAbsoluteError: 0.0594072155654430 | Loss: 0.0065107481712524 | Epoch: 1303 |

MeanAbsoluteError: 0.0645157322287560 | Loss: 0.0071178560520881 | Epoch: 1304 | MeanAbsolute

MeanAbsoluteError: 0.0660323575139046 | Loss: 0.0081524824449662 | Epoch: 1306 |

MeanAbsoluteError: 0.0648837834596634 | Loss: 0.0086707804816494 | Epoch: 1307 | MeanAbsolute

MeanAbsoluteError: 0.0639346539974213 | Loss: 0.0077054121595938 | Epoch: 1309 |

MeanAbsoluteError: 0.0713435336947441 | Loss: 0.0091040640163525 | Epoch: 1310 | MeanAbsolute

MeanAbsoluteError: 0.0657425075769424 | Loss: 0.0076700844155251 | Epoch: 1312 |

MeanAbsoluteError: 0.0635942369699478 | Loss: 0.0079962326862975 | Epoch: 1313 | MeanAbsolute

MeanAbsoluteError: 0.0583355538547039 | Loss: 0.0061331353403511 | Epoch: 1315 |

MeanAbsoluteError: 0.0642955303192139 | Loss: 0.0080147538009138 | Epoch: 1316 | MeanAbsolute

MeanAbsoluteError: 0.0720890760421753 | Loss: 0.0095400127746628 | Epoch: 1318 |

MeanAbsoluteError: 0.0661798492074013 | Loss: 0.0081093414824833 | Epoch: 1319 | MeanAbsolute

MeanAbsoluteError: 0.0621321871876717 | Loss: 0.0072967445766456 | Epoch: 1321 |
```

```
MeanAbsoluteError: 0.0670315921306610 | Loss: 0.0084784864527198 | Epoch: 1322 | MeanAbsolute

MeanAbsoluteError: 0.0638069808483124 | Loss: 0.0081523785462923 | Epoch: 1324 |

MeanAbsoluteError: 0.0657017007470131 | Loss: 0.0083988012816978 | Epoch: 1325 | MeanAbsolute

MeanAbsoluteError: 0.0699718073010445 | Loss: 0.0088471591026367 | Epoch: 1327 |

MeanAbsoluteError: 0.0607786513864994 | Loss: 0.0067464821096049 | Epoch: 1328 | MeanAbsolute

MeanAbsoluteError: 0.0609705150127411 | Loss: 0.0068833680075719 | Epoch: 1330 |

MeanAbsoluteError: 0.0642498433589935 | Loss: 0.0079205412084654 | Epoch: 1331 | MeanAbsolute

MeanAbsoluteError: 0.0650688037276268 | Loss: 0.0084708813252655 | Epoch: 1333 |

MeanAbsoluteError: 0.0654301941394806 | Loss: 0.0077362716695560 | Epoch: 1334 | MeanAbsolute

MeanAbsoluteError: 0.0657037049531937 | Loss: 0.0090830756248882 | Epoch: 1336 |

MeanAbsoluteError: 0.0617531463503838 | Loss: 0.0068937290636359 | Epoch: 1337 | MeanAbsolute

MeanAbsoluteError: 0.0656510069966316 | Loss: 0.0082764692707375 | Epoch: 1339 |

MeanAbsoluteError: 0.0668194517493248 | Loss: 0.0085012014267825 | Epoch: 1340 | MeanAbsolute

MeanAbsoluteError: 0.0643763244152069 | Loss: 0.0076960352947966 | Epoch: 1342 |

MeanAbsoluteError: 0.0693575218319893 | Loss: 0.0088684189810738 | Epoch: 1343 | MeanAbsolute

MeanAbsoluteError: 0.0658741444349289 | Loss: 0.0083490362577625 | Epoch: 1345 |

MeanAbsoluteError: 0.0628336071968079 | Loss: 0.0079287036413371 | Epoch: 1346 | MeanAbsolute

MeanAbsoluteError: 0.0627309381961823 | Loss: 0.0072050519320813 | Epoch: 1348 |

MeanAbsoluteError: 0.0635385215282440 | Loss: 0.0077831743975306 | Epoch: 1349 | MeanAbsolute
```

```
MeanAbsoluteError: 0.0631478875875473 | Loss: 0.0080020195328689 | Epoch: 1351 |
MeanAbsoluteError: 0.0667655616998672 | Loss: 0.0079982629829222 | Epoch: 1352 | MeanAbsolute
MeanAbsoluteError: 0.0596620626747608 | Loss: 0.0065802116856745 | Epoch: 1354 |
MeanAbsoluteError: 0.0697294473648071 | Loss: 0.0091364694974861 | Epoch: 1355 | MeanAbsolute
MeanAbsoluteError: 0.0668593719601631 | Loss: 0.0083080786987315 | Epoch: 1357 |
MeanAbsoluteError: 0.0655754730105400 | Loss: 0.0081913021148891 | Epoch: 1358 | MeanAbsolute
MeanAbsoluteError: 0.0666330382227898 | Loss: 0.0079030964575456 | Epoch: 1360 |
MeanAbsoluteError: 0.0652557387948036 | Loss: 0.0086389265989995 | Epoch: 1361 | MeanAbsolute
MeanAbsoluteError: 0.0639628246426582 | Loss: 0.0072017002135181 | Epoch: 1363 |
MeanAbsoluteError: 0.0707581490278244 | Loss: 0.0087081964027796 | Epoch: 1364 | MeanAbsolute
MeanAbsoluteError: 0.0626756548881531 | Loss: 0.0070143927798199 | Epoch: 1366 |
MeanAbsoluteError: 0.0685416087508202 | Loss: 0.0087187562713128 | Epoch: 1367 | MeanAbsolute
MeanAbsoluteError: 0.0647659748792648 | Loss: 0.0080185976124752 | Epoch: 1369 |
MeanAbsoluteError: 0.0687026232481003 | Loss: 0.0087558793647501 | Epoch: 1370 | MeanAbsolute
MeanAbsoluteError: 0.0628857091069221 | Loss: 0.0075636881236763 | Epoch: 1372 |
MeanAbsoluteError: 0.0630418360233307 | Loss: 0.0075581621971151 | Epoch: 1373 | MeanAbsolute
MeanAbsoluteError: 0.0602940693497658 | Loss: 0.0062444118406953 | Epoch: 1375 |
MeanAbsoluteError: 0.0589975304901600 | Loss: 0.0067708865271485 | Epoch: 1376 | MeanAbsolute
MeanAbsoluteError: 0.0594511032104492 | Loss: 0.0063276568336247 | Epoch: 1378 |
```

```
MeanAbsoluteError: 0.0621821098029613 | Loss: 0.0071327639724192 | Epoch: 1379 | MeanAbsolute

MeanAbsoluteError: 0.0652953088283539 | Loss: 0.0074444382189752 | Epoch: 1381 |

MeanAbsoluteError: 0.0618769824504852 | Loss: 0.0072001051944001 | Epoch: 1382 | MeanAbsolute

MeanAbsoluteError: 0.0619668141007423 | Loss: 0.0073720962977245 | Epoch: 1384 |

MeanAbsoluteError: 0.0641926154494286 | Loss: 0.0079162271381574 | Epoch: 1385 | MeanAbsolute

MeanAbsoluteError: 0.0603689886629581 | Loss: 0.0067396732764367 | Epoch: 1387 |

MeanAbsoluteError: 0.0684203356504440 | Loss: 0.0088764059687189 | Epoch: 1388 | MeanAbsolute

MeanAbsoluteError: 0.0608590431511402 | Loss: 0.0071028358387897 | Epoch: 1390 |

MeanAbsoluteError: 0.0610859282314777 | Loss: 0.0070337120117862 | Epoch: 1391 | MeanAbsolute

MeanAbsoluteError: 0.0631525143980980 | Loss: 0.0066375193205507 | Epoch: 1393 |

MeanAbsoluteError: 0.0659789368510246 | Loss: 0.0086677836634590 | Epoch: 1394 | MeanAbsolute

MeanAbsoluteError: 0.0585643835365772 | Loss: 0.0069732424174920 | Epoch: 1396 |

MeanAbsoluteError: 0.0650768503546715 | Loss: 0.0075330008691526 | Epoch: 1397 | MeanAbsolute

MeanAbsoluteError: 0.0665219947695732 | Loss: 0.0074907748852759 | Epoch: 1399 |

MeanAbsoluteError: 0.0664467960596085 | Loss: 0.0077286590708779 | Epoch: 1400 | MeanAbsolute

MeanAbsoluteError: 0.0638523325324059 | Loss: 0.0079040845132962 | Epoch: 1402 |

MeanAbsoluteError: 0.0660431310534477 | Loss: 0.0087874914874798 | Epoch: 1403 | MeanAbsolute

MeanAbsoluteError: 0.0609945282340050 | Loss: 0.0065798147047462 | Epoch: 1405 |

MeanAbsoluteError: 0.0624893382191658 | Loss: 0.0076832239791353 | Epoch: 1406 | MeanAbsolute
```

```
MeanAbsoluteError: 0.0605614073574543 | Loss: 0.0074499432366914 | Epoch: 1408 |
MeanAbsoluteError: 0.0696291700005531 | Loss: 0.0093638487560141 | Epoch: 1409 | MeanAbsolute
MeanAbsoluteError: 0.0643391385674477 | Loss: 0.0076526276814305 | Epoch: 1411 |
MeanAbsoluteError: 0.0631081163883209 | Loss: 0.0070025545966807 | Epoch: 1412 | MeanAbsolute
MeanAbsoluteError: 0.0632226914167404 | Loss: 0.0072618221079271 | Epoch: 1414 |
MeanAbsoluteError: 0.0627158582210541 | Loss: 0.0079367169177567 | Epoch: 1415 | MeanAbsolute
MeanAbsoluteError: 0.0725618675351143 | Loss: 0.0099599302406326 | Epoch: 1417 |
MeanAbsoluteError: 0.0609041899442673 | Loss: 0.0071207839619122 | Epoch: 1418 | MeanAbsolute
MeanAbsoluteError: 0.0654263272881508 | Loss: 0.0078905881053894 | Epoch: 1420 |
MeanAbsoluteError: 0.0647323727607727 | Loss: 0.0078630630763413 | Epoch: 1421 | MeanAbsolute
MeanAbsoluteError: 0.0633253753185272 | Loss: 0.0072540082567897 | Epoch: 1423 |
MeanAbsoluteError: 0.0648477971553802 | Loss: 0.0079165506193840 | Epoch: 1424 | MeanAbsolute
MeanAbsoluteError: 0.0595656074583530 | Loss: 0.0065818863581626 | Epoch: 1426 |
MeanAbsoluteError: 0.0601306669414043 | Loss: 0.0072479626874216 | Epoch: 1427 | MeanAbsolute
MeanAbsoluteError: 0.0632165223360062 | Loss: 0.0068992029945851 | Epoch: 1429 |
MeanAbsoluteError: 0.0624227188527584 | Loss: 0.0068718473607078 | Epoch: 1430 | MeanAbsolute
MeanAbsoluteError: 0.0601941756904125 | Loss: 0.0070093692820592 | Epoch: 1432 |
MeanAbsoluteError: 0.0636248141527176 | Loss: 0.0078177529574047 | Epoch: 1433 | MeanAbsolute
MeanAbsoluteError: 0.0616594441235065 | Loss: 0.0070151825618571 | Epoch: 1435 |
```

```
MeanAbsoluteError: 0.0656498223543167 | Loss: 0.0080918291372169 | Epoch: 1436 | MeanAbsolute

MeanAbsoluteError: 0.0588392131030560 | Loss: 0.0067577041208945 | Epoch: 1438 |

MeanAbsoluteError: 0.0636597648262978 | Loss: 0.0073712377665894 | Epoch: 1439 | MeanAbsolute

MeanAbsoluteError: 0.0631179288029671 | Loss: 0.0080914937816427 | Epoch: 1441 |

MeanAbsoluteError: 0.0628407523036003 | Loss: 0.0071765589200000 | Epoch: 1442 | MeanAbsolute
Returned to Spot: Validation loss: 0.008217219537327765

config: {'_L_in': 10, '_L_out': 1, 'l1': 16, 'dropout_prob': 0.1773189149831582, 'lr_mult': 9
Epoch: 1 | MeanAbsoluteError: 0.1428941190242767 | Loss: 0.0298500930269559 | Epoch: 2 |

MeanAbsoluteError: 0.1393741071224213 | Loss: 0.0298705764984091 | Epoch: 3 | MeanAbsoluteErr

MeanAbsoluteError: 0.1410290896892548 | Loss: 0.0292373106845965 | Returned to Spot: Validati

config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.3840970624671163, 'lr_mult': 4
Epoch: 1 | MeanAbsoluteError: 0.1503297239542007 | Loss: 0.0364762070637785 | Epoch: 2 | Mean

MeanAbsoluteError: 0.1151882186532021 | Loss: 0.0212563546129355 | Epoch: 8 | MeanAbsoluteErr

MeanAbsoluteError: 0.0954033732414246 | Loss: 0.0148606114089489 | Epoch: 12 | MeanAbsoluteE

MeanAbsoluteError: 0.0820449739694595 | Loss: 0.0122394334635158 | Epoch: 19 | MeanAbsoluteE

MeanAbsoluteError: 0.0790838301181793 | Loss: 0.0107780087059119 | Epoch: 23 | MeanAbsoluteE

MeanAbsoluteError: 0.0798358321189880 | Loss: 0.0110917718848214 | Epoch: 30 | MeanAbsoluteE

MeanAbsoluteError: 0.0742576047778130 | Loss: 0.0099593943389329 | Epoch: 34 | MeanAbsoluteE

MeanAbsoluteError: 0.0659698098897934 | Loss: 0.0079800440458042 | Epoch: 41 | MeanAbsoluteE

MeanAbsoluteError: 0.0806830376386642 | Loss: 0.0113666646617563 | Epoch: 45 | MeanAbsoluteE

MeanAbsoluteError: 0.0648790001869202 | Loss: 0.0081240745065244 | Epoch: 52 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.0632500573992729 | Loss: 0.0082429487589060 | Epoch: 56 | MeanAbsoluteEr

MeanAbsoluteError: 0.0648432075977325 | Loss: 0.0073641805731887 | Epoch: 63 | MeanAbsoluteEr

MeanAbsoluteError: 0.0647667571902275 | Loss: 0.0082157201495169 | Epoch: 67 | MeanAbsoluteEr

MeanAbsoluteError: 0.0600982233881950 | Loss: 0.0068770791465220 | Epoch: 74 | MeanAbsoluteEr

MeanAbsoluteError: 0.0976491570472717 | Loss: 0.0150856927380358 | Epoch: 78 | MeanAbsoluteEr

MeanAbsoluteError: 0.0677435994148254 | Loss: 0.0077096135539346 | Epoch: 85 | MeanAbsoluteEr

MeanAbsoluteError: 0.0580953210592270 | Loss: 0.0072747969886558 | Epoch: 89 | MeanAbsoluteEr

MeanAbsoluteError: 0.0650054886937141 | Loss: 0.0074428268279390 | Epoch: 96 | MeanAbsoluteEr

MeanAbsoluteError: 0.0616717264056206 | Loss: 0.0080109845179035 | Epoch: 100 | MeanAbsolute

MeanAbsoluteError: 0.0568202398717403 | Loss: 0.0064301671610385 | Epoch: 107 | MeanAbsolute

MeanAbsoluteError: 0.0645977929234505 | Loss: 0.0083148744551612 | Epoch: 111 | MeanAbsolute

MeanAbsoluteError: 0.0724008381366730 | Loss: 0.0089462961462375 | Epoch: 118 | MeanAbsolute

MeanAbsoluteError: 0.0609829761087894 | Loss: 0.0069774485095159 | Epoch: 122 | MeanAbsolute

MeanAbsoluteError: 0.0565027110278606 | Loss: 0.0064870874477109 | Epoch: 129 | MeanAbsolute

MeanAbsoluteError: 0.0572304353117943 | Loss: 0.0062431189611456 | Epoch: 133 | MeanAbsolute

MeanAbsoluteError: 0.0584750249981880 | Loss: 0.0069484745663909 | Epoch: 140 | MeanAbsolute

MeanAbsoluteError: 0.0618731006979942 | Loss: 0.0066643518604044 | Epoch: 144 | MeanAbsolute

MeanAbsoluteError: 0.0625421926379204 | Loss: 0.0077170557710097 | Epoch: 151 | MeanAbsolute

MeanAbsoluteError: 0.0541630610823631 | Loss: 0.0060775161347058 | Epoch: 155 | MeanAbsolute
```

```
MeanAbsoluteError: 0.0572330914437771 | Loss: 0.0066573909686045 | Epoch: 162 | MeanAbsoluteE

MeanAbsoluteError: 0.0588993579149246 | Loss: 0.0066544464666789 | Epoch: 166 | MeanAbsoluteE

MeanAbsoluteError: 0.0651962906122208 | Loss: 0.0078206826108630 | Epoch: 173 | MeanAbsoluteE

MeanAbsoluteError: 0.0599686168134212 | Loss: 0.0061422380232742 | Epoch: 177 | MeanAbsoluteE

MeanAbsoluteError: 0.0560736097395420 | Loss: 0.0061824691089752 | Epoch: 184 | MeanAbsoluteE

MeanAbsoluteError: 0.0629627332091331 | Loss: 0.0072708878494603 | Epoch: 188 | MeanAbsoluteE

MeanAbsoluteError: 0.0614710822701454 | Loss: 0.0074393066323338 | Epoch: 195 | MeanAbsoluteE

MeanAbsoluteError: 0.0606770329177380 | Loss: 0.0068628632151032 | Epoch: 199 | MeanAbsoluteE

MeanAbsoluteError: 0.0610444843769073 | Loss: 0.0071070607304328 | Epoch: 206 | MeanAbsoluteE

MeanAbsoluteError: 0.0577037297189236 | Loss: 0.0064703127718531 | Epoch: 210 | MeanAbsoluteE

MeanAbsoluteError: 0.0567633248865604 | Loss: 0.0060802324334968 | Epoch: 217 | MeanAbsoluteE

MeanAbsoluteError: 0.0557916723191738 | Loss: 0.0064796283075243 | Epoch: 221 | MeanAbsoluteE

MeanAbsoluteError: 0.0641929656267166 | Loss: 0.0080433753870526 | Epoch: 228 | MeanAbsoluteE

MeanAbsoluteError: 0.0608983114361763 | Loss: 0.0067103956152048 | Epoch: 232 | MeanAbsoluteE

MeanAbsoluteError: 0.0585256069898605 | Loss: 0.0073398345389011 | Early stopping at epoch 23
Returned to Spot: Validation loss: 0.007339834538901127


config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.20883473758270768, 'lr_mult':
Epoch: 1 | MeanAbsoluteError: 0.2607739567756653 | Loss: 0.0916581896967009 | Epoch: 2 | Mean

MeanAbsoluteError: 0.0683658123016357 | Loss: 0.0078910936625968 | Epoch: 20 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.0526944920420647 | Loss: 0.0048527442666359 | Epoch: 39 | MeanAbsoluteEr

MeanAbsoluteError: 0.0554284490644932 | Loss: 0.0048665433484865 | Epoch: 58 | MeanAbsoluteEr

MeanAbsoluteError: 0.0520139858126640 | Loss: 0.0047195544343834 | Epoch: 77 | MeanAbsoluteEr

MeanAbsoluteError: 0.0498494543135166 | Loss: 0.0040104242262283 | Epoch: 96 | MeanAbsoluteEr
Returned to Spot: Validation loss: 0.002804433338736233

spotPython tuning: 0.002804433338736233 [----------] 2.72%


config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.2303248786572344, 'lr_mult': 1
Epoch: 1 | MeanAbsoluteError: 0.1530535817146301 | Loss: 0.0366678338890013 | Epoch: 2 | Mean

MeanAbsoluteError: 0.2358370721340179 | Loss: 0.0646486376461230 | Epoch: 20 | MeanAbsoluteEr

MeanAbsoluteError: 0.0867798700928688 | Loss: 0.0117060865863766 | Epoch: 39 | MeanAbsoluteEr

MeanAbsoluteError: 0.0963144153356552 | Loss: 0.0116625842393229 | Epoch: 58 | MeanAbsoluteEr

MeanAbsoluteError: 0.0790779367089272 | Loss: 0.0076163382220425 | Epoch: 77 | MeanAbsoluteEr

MeanAbsoluteError: 0.0479007735848427 | Loss: 0.0041397881363273 | Epoch: 96 | MeanAbsoluteEr

MeanAbsoluteError: 0.0398120358586311 | Loss: 0.0025625563300166 | Epoch: 115 | MeanAbsoluteE
Returned to Spot: Validation loss: 0.006521587468389618

spotPython tuning: 0.002804433338736233 [#---------] 5.92%


config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.1375889995647968, 'lr_mult': 4
Epoch: 1 | MeanAbsoluteError: 0.1716033071279526 | Loss: 0.0459190089451639 | Epoch: 2 | Mean

MeanAbsoluteError: 0.1055722683668137 | Loss: 0.0149187247121805 | Epoch: 20 | MeanAbsoluteEr

MeanAbsoluteError: 0.0399528294801712 | Loss: 0.0032751430377462 | Epoch: 39 | MeanAbsoluteEr
```

```
MeanAbsoluteError: 0.0379076190292835 | Loss: 0.0023372049296373 | Epoch: 58 | MeanAbsoluteEr

MeanAbsoluteError: 0.0355779193341732 | Loss: 0.0025493200028952 | Epoch: 77 | MeanAbsoluteEr

MeanAbsoluteError: 0.0714666843414307 | Loss: 0.0064367749605720 | Epoch: 96 | MeanAbsoluteEr
Returned to Spot: Validation loss: 0.007592509966343641

spotPython tuning: 0.002804433338736233 [#---------] 8.64%


config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.6395024394785109, 'lr_mult': 9
Epoch: 1 | MeanAbsoluteError: 0.1518076807260513 | Loss: 0.0358457343750878 | Epoch: 2 | Mean

MeanAbsoluteError: 0.1282563805580139 | Loss: 0.0266165558346792 | Epoch: 20 | MeanAbsoluteEr

MeanAbsoluteError: 0.1457662135362625 | Loss: 0.0317250048545630 | Epoch: 39 | MeanAbsoluteEr

MeanAbsoluteError: 0.1081419512629509 | Loss: 0.0202012162557558 | Epoch: 58 | MeanAbsoluteEr

MeanAbsoluteError: 0.1247476562857628 | Loss: 0.0237786326263296 | Epoch: 77 | MeanAbsoluteEr

MeanAbsoluteError: 0.0971428826451302 | Loss: 0.0183067365402454 | Epoch: 96 | MeanAbsoluteEr

MeanAbsoluteError: 0.1243703365325928 | Loss: 0.0252878360548302 | Epoch: 115 | MeanAbsoluteE
Returned to Spot: Validation loss: 0.02182327902042552
spotPython tuning: 0.002804433338736233 [#---------] 11.38%


config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.22575294745401697, 'lr_mult':
Epoch: 1 | MeanAbsoluteError: 0.2965242266654968 | Loss: 0.1173508214323144 | Epoch: 2 | Mean

MeanAbsoluteError: 0.1938448846340179 | Loss: 0.0447851936087797 | Epoch: 20 | MeanAbsoluteEr

MeanAbsoluteError: 0.1442598253488541 | Loss: 0.0246203210400908 | Epoch: 39 | MeanAbsoluteEr

MeanAbsoluteError: 0.1086758300662041 | Loss: 0.0153474166842276 | Epoch: 58 | MeanAbsoluteEr

MeanAbsoluteError: 0.1397401243448257 | Loss: 0.0237558229189170 | Epoch: 77 | MeanAbsoluteEr
```

```
MeanAbsoluteError: 0.0816402509808540 | Loss: 0.0110994414426386 | Epoch: 96 | MeanAbsoluteEr

MeanAbsoluteError: 0.0634299665689468 | Loss: 0.0057553035991364 | Epoch: 115 | MeanAbsoluteE
Returned to Spot: Validation loss: 0.010810166352281445

spotPython tuning: 0.002804433338736233 [#---------] 14.76%


config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.41398397266914844, 'lr_mult':
Epoch: 1 | MeanAbsoluteError: 0.2596502602100372 | Loss: 0.0945879371934815 | Epoch: 2 | Mean

MeanAbsoluteError: 0.1320084184408188 | Loss: 0.0236712646866707 | Epoch: 20 | MeanAbsoluteEr

MeanAbsoluteError: 0.0850442126393318 | Loss: 0.0108315610130759 | Epoch: 39 | MeanAbsoluteEr

MeanAbsoluteError: 0.0804186537861824 | Loss: 0.0115741543520830 | Epoch: 58 | MeanAbsoluteEr

MeanAbsoluteError: 0.0983642414212227 | Loss: 0.0141558545445533 | Epoch: 77 | MeanAbsoluteEr

MeanAbsoluteError: 0.1126966699957848 | Loss: 0.0172907811050352 | Epoch: 96 | MeanAbsoluteEr

MeanAbsoluteError: 0.0466120727360249 | Loss: 0.0047214235681550 | Epoch: 115 | MeanAbsoluteE

MeanAbsoluteError: 0.0771155878901482 | Loss: 0.0087268516225250 | Epoch: 134 | MeanAbsoluteE
Returned to Spot: Validation loss: 0.01561069400294831
spotPython tuning: 0.002804433338736233 [##--------] 17.91%


config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.13138748199303163, 'lr_mult':
Epoch: 1 | MeanAbsoluteError: 0.1409188508987427 | Loss: 0.0317683588027170 | Epoch: 2 | Mean

MeanAbsoluteError: 0.0624709315598011 | Loss: 0.0070033554281843 | Epoch: 20 | MeanAbsoluteEr

MeanAbsoluteError: 0.0627640187740326 | Loss: 0.0056000542704408 | Epoch: 39 | MeanAbsoluteEr

MeanAbsoluteError: 0.1079071536660194 | Loss: 0.0154208948002442 | Epoch: 58 | MeanAbsoluteEr

MeanAbsoluteError: 0.0414252690970898 | Loss: 0.0033665330051199 | Epoch: 77 | MeanAbsoluteEr
```

```
MeanAbsoluteError: 0.0520555600523949 | Loss: 0.0047131366415047 | Epoch: 96 | MeanAbsoluteEr

MeanAbsoluteError: 0.0455248244106770 | Loss: 0.0030320072975500 | Epoch: 115 | MeanAbsoluteE
Returned to Spot: Validation loss: 0.0022381137027160116

spotPython tuning: 0.0022381137027160116 [##--------] 20.85%


config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.3843019526408822, 'lr_mult': 4
Epoch: 1 | MeanAbsoluteError: 0.1445990055799484 | Loss: 0.0337781555533997 | Epoch: 2 | Mean

MeanAbsoluteError: 0.0896276012063026 | Loss: 0.0134052002471627 | Epoch: 12 | MeanAbsoluteEr

MeanAbsoluteError: 0.0615290515124798 | Loss: 0.0076470190481479 | Epoch: 23 | MeanAbsoluteEr

MeanAbsoluteError: 0.0622271485626698 | Loss: 0.0074696385448700 | Epoch: 34 | MeanAbsoluteEr

MeanAbsoluteError: 0.0496069416403770 | Loss: 0.0049578724704732 | Epoch: 45 | MeanAbsoluteEr

MeanAbsoluteError: 0.0543858259916306 | Loss: 0.0055520063278412 | Epoch: 56 | MeanAbsoluteEr

MeanAbsoluteError: 0.0572311915457249 | Loss: 0.0056233402942062 | Epoch: 67 | MeanAbsoluteEr

MeanAbsoluteError: 0.0585619434714317 | Loss: 0.0057607427925656 | Epoch: 78 | MeanAbsoluteEr

MeanAbsoluteError: 0.0471083559095860 | Loss: 0.0043308221410323 | Epoch: 89 | MeanAbsoluteEr

MeanAbsoluteError: 0.0641004666686058 | Loss: 0.0070295918420398 | Epoch: 100 | MeanAbsoluteE

MeanAbsoluteError: 0.0456455610692501 | Loss: 0.0047866025875249 | Epoch: 111 | MeanAbsoluteE

MeanAbsoluteError: 0.0523261576890945 | Loss: 0.0055197405561462 | Epoch: 122 | MeanAbsoluteE

MeanAbsoluteError: 0.0540135651826859 | Loss: 0.0054688645391970 | Epoch: 133 | MeanAbsoluteE

MeanAbsoluteError: 0.0522469803690910 | Loss: 0.0051550298812799 | Epoch: 144 | MeanAbsoluteE
Returned to Spot: Validation loss: 0.006224949749211143
```

```
spotPython tuning: 0.0022381137027160116 [###-------] 26.68%


config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.06224079918823905, 'lr_mult':
Epoch: 1 | MeanAbsoluteError: 0.2304285466670990 | Loss: 0.0701870630054097 | Epoch: 2 | Mea

MeanAbsoluteError: 0.0542467832565308 | Loss: 0.0046276009188181 | Epoch: 20 | MeanAbsoluteE

MeanAbsoluteError: 0.0551394522190094 | Loss: 0.0056286754062105 | Epoch: 39 | MeanAbsoluteE

MeanAbsoluteError: 0.0444856770336628 | Loss: 0.0036592895508205 | Epoch: 58 | MeanAbsoluteE

MeanAbsoluteError: 0.0470491088926792 | Loss: 0.0028254594837659 | Epoch: 77 | MeanAbsoluteE

MeanAbsoluteError: 0.0310134626924992 | Loss: 0.0016074419254437 | Epoch: 96 | MeanAbsoluteE

MeanAbsoluteError: 0.0772935450077057 | Loss: 0.0079576304231427 | Epoch: 115 | MeanAbsolute
Returned to Spot: Validation loss: 0.0017883027888417832

spotPython tuning: 0.0017883027888417832 [###-------] 30.01%


config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.003312242887064156, 'lr_mult'
Epoch: 1 | MeanAbsoluteError: 0.1838473677635193 | Loss: 0.0470597580271332 | Epoch: 2 | Mea

spotPython tuning: 0.0017883027888417832 [###-------] 31.21%


config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.06710968770738812, 'lr_mult':
Epoch: 1 | MeanAbsoluteError: 0.1467798352241516 | Loss: 0.0330562819971850 | Epoch: 2 | Mea

MeanAbsoluteError: 0.0886431261897087 | Loss: 0.0117270095322869 | Epoch: 20 | MeanAbsoluteE

MeanAbsoluteError: 0.1729440689086914 | Loss: 0.0330319266374174 | Epoch: 39 | MeanAbsoluteE

MeanAbsoluteError: 0.0780751332640648 | Loss: 0.0087440857957853 | Epoch: 58 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.0510008297860622 | Loss: 0.0037116099724938 | Epoch: 77 | MeanAbsoluteEr
Returned to Spot: Validation loss: 0.0021288082444746244

spotPython tuning: 0.0017883027888417832 [###-------] 34.12%


config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.21454831994090667, 'lr_mult':
Epoch: 1 | MeanAbsoluteError: 0.1428184956312180 | Loss: 0.0317707500469528 | Epoch: 2 | Mean

MeanAbsoluteError: 0.0721909850835800 | Loss: 0.0087734185967987 | Epoch: 20 | MeanAbsoluteEr

MeanAbsoluteError: 0.0457966327667236 | Loss: 0.0044145906730065 | Epoch: 39 | MeanAbsoluteEr

MeanAbsoluteError: 0.0787620246410370 | Loss: 0.0081594361629533 | Epoch: 58 | MeanAbsoluteEr

MeanAbsoluteError: 0.0447855629026890 | Loss: 0.0034129199890518 | Epoch: 77 | MeanAbsoluteEr

MeanAbsoluteError: 0.0371668487787247 | Loss: 0.0026646649930626 | Epoch: 96 | MeanAbsoluteEr

MeanAbsoluteError: 0.0439669825136662 | Loss: 0.0034946575944655 | Epoch: 115 | MeanAbsoluteE

MeanAbsoluteError: 0.0549341067671776 | Loss: 0.0054072394084774 | Epoch: 134 | MeanAbsoluteE
Returned to Spot: Validation loss: 0.004670502305471976

spotPython tuning: 0.0017883027888417832 [####------] 37.66%


config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.0, 'lr_mult': 6.4132776360446
Epoch: 1 | MeanAbsoluteError: 0.1555278301239014 | Loss: 0.0347338503373689 | Epoch: 2 | Mean

MeanAbsoluteError: 0.0620208866894245 | Loss: 0.0060892082616001 | Epoch: 13 | MeanAbsoluteEr

MeanAbsoluteError: 0.0727357044816017 | Loss: 0.0069571955154013 | Epoch: 25 | MeanAbsoluteEr

MeanAbsoluteError: 0.0704571604728699 | Loss: 0.0051375382922982 | Epoch: 37 | MeanAbsoluteEr

MeanAbsoluteError: 0.0447442345321178 | Loss: 0.0024552654986588 | Epoch: 49 | MeanAbsoluteEr
```

```
MeanAbsoluteError: 0.0153476176783442 | Loss: 0.0003692856518778 | Epoch: 61 | MeanAbsoluteE

MeanAbsoluteError: 0.0283896978944540 | Loss: 0.0012179083718557 | Epoch: 73 | MeanAbsoluteE

MeanAbsoluteError: 0.0157770253717899 | Loss: 0.0004216783368304 | Epoch: 85 | MeanAbsoluteE

MeanAbsoluteError: 0.0326439775526524 | Loss: 0.0014554828774539 | Epoch: 97 | MeanAbsoluteE

MeanAbsoluteError: 0.0290566757321358 | Loss: 0.0011358285272237 | Epoch: 109 | MeanAbsolute

MeanAbsoluteError: 0.0640832111239433 | Loss: 0.0043895197664633 | Epoch: 121 | MeanAbsolute
Returned to Spot: Validation loss: 0.003406676662914259

spotPython tuning: 0.0017883027888417832 [####------] 43.99%


config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.06308103865735404, 'lr_mult':
Epoch: 1 | MeanAbsoluteError: 0.2582009732723236 | Loss: 0.0832594327236477 | Epoch: 2 | Mea

MeanAbsoluteError: 0.0703782364726067 | Loss: 0.0069032661053107 | Epoch: 20 | MeanAbsoluteE

MeanAbsoluteError: 0.0536159202456474 | Loss: 0.0045820868476049 | Epoch: 39 | MeanAbsoluteE

MeanAbsoluteError: 0.0995748266577721 | Loss: 0.0122388745658100 | Epoch: 58 | MeanAbsoluteE

MeanAbsoluteError: 0.0358383990824223 | Loss: 0.0024892071194604 | Early stopping at epoch 75
Returned to Spot: Validation loss: 0.0024892071194603646

spotPython tuning: 0.0017883027888417832 [#####-----] 49.25%


config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.17701715022569864, 'lr_mult':
Epoch: 1 | MeanAbsoluteError: 0.1818190813064575 | Loss: 0.0476691595425731 | Epoch: 2 | Mea

MeanAbsoluteError: 0.0686335265636444 | Loss: 0.0080861086761089 | Epoch: 20 | MeanAbsoluteE

MeanAbsoluteError: 0.1130295470356941 | Loss: 0.0173505815236192 | Epoch: 39 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.0370396785438061 | Loss: 0.0023239121380213 | Epoch: 58 | MeanAbsoluteEr
Returned to Spot: Validation loss: 0.003953154676740891

spotPython tuning: 0.0017883027888417832 [#####-----] 53.86%


config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.037804400078849484, 'lr_mult'
Epoch: 1 | MeanAbsoluteError: 0.1070512384176254 | Loss: 0.0186644898433434 | Epoch: 2 | Mean

MeanAbsoluteError: 0.0862432792782784 | Loss: 0.0109361101334032 | Epoch: 20 | MeanAbsoluteEr

MeanAbsoluteError: 0.0823937207460403 | Loss: 0.0081055790038877 | Epoch: 39 | MeanAbsoluteEr

MeanAbsoluteError: 0.0510830804705620 | Loss: 0.0031247071946334 | Epoch: 58 | MeanAbsoluteEr

MeanAbsoluteError: 0.0189086906611919 | Loss: 0.0006789836058344 | Epoch: 77 | MeanAbsoluteEr

MeanAbsoluteError: 0.0696948990225792 | Loss: 0.0061853133436096 | Epoch: 96 | MeanAbsoluteEr

MeanAbsoluteError: 0.0492193885147572 | Loss: 0.0037069686161550 | Epoch: 115 | MeanAbsolutel

spotPython tuning: 0.0017883027888417832 [######----] 59.57%


config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.07335289513353108, 'lr_mult':
Epoch: 1 | MeanAbsoluteError: 0.1271923631429672 | Loss: 0.0257343424876270 | Epoch: 2 | Mean

MeanAbsoluteError: 0.0947592630982399 | Loss: 0.0112891769477803 | Epoch: 20 | MeanAbsoluteEr

MeanAbsoluteError: 0.1028797030448914 | Loss: 0.0130914233036731 | Epoch: 39 | MeanAbsoluteEr

MeanAbsoluteError: 0.0524498596787453 | Loss: 0.0047239248922683 | Epoch: 58 | MeanAbsoluteEr

MeanAbsoluteError: 0.0373034402728081 | Loss: 0.0027142577669828 | Epoch: 77 | MeanAbsoluteEr

MeanAbsoluteError: 0.0428089387714863 | Loss: 0.0031631798452796 | Epoch: 96 | MeanAbsoluteEr

MeanAbsoluteError: 0.0545675083994865 | Loss: 0.0046337389534241 | Epoch: 115 | MeanAbsolutel
```

```
spotPython tuning: 0.0017883027888417832 [######----] 65.00%


config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.11350519116598666, 'lr_mult':
Epoch: 1 | MeanAbsoluteError: 0.1866738200187683 | Loss: 0.0509602447089396 | Epoch: 2 | Mean

MeanAbsoluteError: 0.0486808046698570 | Loss: 0.0040204955608045 | Epoch: 20 | MeanAbsoluteE

MeanAbsoluteError: 0.0899174287915230 | Loss: 0.0123622885807172 | Epoch: 39 | MeanAbsoluteE

MeanAbsoluteError: 0.0484480969607830 | Loss: 0.0042866503537976 | Epoch: 58 | MeanAbsoluteE

MeanAbsoluteError: 0.0444098599255085 | Loss: 0.0036493282532319 | Epoch: 77 | MeanAbsoluteE

MeanAbsoluteError: 0.0402500145137310 | Loss: 0.0024821820521825 | Epoch: 96 | MeanAbsoluteE

MeanAbsoluteError: 0.0367868356406689 | Loss: 0.0023086552207946 | Epoch: 115 | MeanAbsolute

MeanAbsoluteError: 0.0889655798673630 | Loss: 0.0101839388524623 | Epoch: 134 | MeanAbsolute

MeanAbsoluteError: 0.0315490551292896 | Loss: 0.0016118195735065 | Epoch: 153 | MeanAbsolute

MeanAbsoluteError: 0.0626941323280334 | Loss: 0.0057627130422349 | Epoch: 172 | MeanAbsolute

MeanAbsoluteError: 0.0433305427432060 | Loss: 0.0035683037223000 | Epoch: 191 | MeanAbsolute

MeanAbsoluteError: 0.0440207868814468 | Loss: 0.0029337752034496 | Epoch: 210 | MeanAbsolute

MeanAbsoluteError: 0.0665543749928474 | Loss: 0.0062622528302630 | Epoch: 229 | MeanAbsolute

MeanAbsoluteError: 0.0268750600516796 | Loss: 0.0013822300319780 | Epoch: 248 | MeanAbsolute

MeanAbsoluteError: 0.0495472960174084 | Loss: 0.0053523366192454 | Epoch: 267 | MeanAbsolute

MeanAbsoluteError: 0.0397260934114456 | Loss: 0.0026488682794336 | Epoch: 286 | MeanAbsolute

MeanAbsoluteError: 0.0318978168070316 | Loss: 0.0018999703001770 | Epoch: 305 | MeanAbsolute
```

```
MeanAbsoluteError: 0.0337230004370213 | Loss: 0.0023211997100397 | Epoch: 324 | MeanAbsoluteE
Returned to Spot: Validation loss: 0.008799069021877489

spotPython tuning: 0.0017883027888417832 [#######---] 74.24%


config: {'_L_in': 10, '_L_out': 1, 'l1': 64, 'dropout_prob': 0.0, 'lr_mult': 6.3102879422147
Epoch: 1 | MeanAbsoluteError: 0.1234666407108307 | Loss: 0.0242232225443188 | Epoch: 2 | Mean

MeanAbsoluteError: 0.0723460987210274 | Loss: 0.0080969637063773 | Epoch: 21 | MeanAbsoluteE

MeanAbsoluteError: 0.0376422591507435 | Loss: 0.0021731389968313 | Epoch: 41 | MeanAbsoluteE

MeanAbsoluteError: 0.0269659608602524 | Loss: 0.0011068804704241 | Epoch: 61 | MeanAbsoluteE
Returned to Spot: Validation loss: 0.008449405284696504

spotPython tuning: 0.0017883027888417832 [########--] 79.32%


config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.006809635108859902, 'lr_mult'
Epoch: 1 | MeanAbsoluteError: 0.2515684366226196 | Loss: 0.0823533574217244 | Epoch: 2 | Mean

MeanAbsoluteError: 0.0663642361760139 | Loss: 0.0070326886335878 | Epoch: 20 | MeanAbsoluteE

MeanAbsoluteError: 0.0757663100957870 | Loss: 0.0078976368776670 | Epoch: 39 | MeanAbsoluteE

MeanAbsoluteError: 0.0472108572721481 | Loss: 0.0041947351878901 | Epoch: 58 | MeanAbsoluteE

MeanAbsoluteError: 0.0483694113790989 | Loss: 0.0029027373407428 | Epoch: 77 | MeanAbsoluteE

MeanAbsoluteError: 0.0369507111608982 | Loss: 0.0017310222488289 | Epoch: 96 | MeanAbsoluteE

MeanAbsoluteError: 0.0215785037726164 | Loss: 0.0008717372900145 | Epoch: 115 | MeanAbsoluteE
Returned to Spot: Validation loss: 0.003090115257930991

spotPython tuning: 0.0017883027888417832 [#########-] 87.18%
```

```
config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.0014688488874371952, 'lr_mult
Epoch: 1 | MeanAbsoluteError: 0.1693249344825745 | Loss: 0.0398516748100519 | Epoch: 2 | Mea

MeanAbsoluteError: 0.0585358813405037 | Loss: 0.0054018746473287 | Epoch: 20 | MeanAbsoluteE

MeanAbsoluteError: 0.0712879821658134 | Loss: 0.0079535811918935 | Epoch: 39 | MeanAbsoluteE

MeanAbsoluteError: 0.0701330825686455 | Loss: 0.0073952527814790 | Epoch: 58 | MeanAbsoluteE

MeanAbsoluteError: 0.0603009983897209 | Loss: 0.0039567824355082 | Epoch: 77 | MeanAbsoluteE
Returned to Spot: Validation loss: 0.0014428024143470746

spotPython tuning: 0.0014428024143470746 [#########-] 94.09%


config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.020556374027351852, 'lr_mult'
Epoch: 1 | MeanAbsoluteError: 0.1312732398509979 | Loss: 0.0274561676932009 | Epoch: 2 | Mea

MeanAbsoluteError: 0.0503463447093964 | Loss: 0.0043180903762971 | Epoch: 20 | MeanAbsoluteE

MeanAbsoluteError: 0.0469813905656338 | Loss: 0.0028534178303457 | Epoch: 39 | MeanAbsoluteE

MeanAbsoluteError: 0.1079115271568298 | Loss: 0.0124806588221537 | Epoch: 58 | MeanAbsoluteE

MeanAbsoluteError: 0.0722857043147087 | Loss: 0.0061927857857786 | Epoch: 77 | MeanAbsoluteE

MeanAbsoluteError: 0.0600162371993065 | Loss: 0.0044217935493706 | Epoch: 96 | MeanAbsoluteE

MeanAbsoluteError: 0.0890959203243256 | Loss: 0.0104360908367916 | Epoch: 115 | MeanAbsolute

MeanAbsoluteError: 0.0262555405497551 | Loss: 0.0010412898922560 | Epoch: 134 | MeanAbsolute

MeanAbsoluteError: 0.0607095733284950 | Loss: 0.0048549266679114 | Epoch: 153 | MeanAbsolute
Returned to Spot: Validation loss: 0.000418222916778177

spotPython tuning: 0.000418222916778177 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x16dee7280>
```

24.9 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section 19.9, see also the description in the documentation: Tensorboard.

24.10 Step 10: Results

After the hyperparameter tuning run is finished, the results can be analyzed as described in Section 19.10.

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

Figure 24.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

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

```
| name          | type    | default    |   lower |   upper |                tuned | transform
|---------------|---------|------------|---------|---------|----------------------|-------------
| _L_in         | int     | 10         |    10.0 |    10.0 |                 10.0 | None
| _L_out        | int     | 1          |     1.0 |     1.0 |                  1.0 | None
| l1            | int     | 3          |     3.0 |     8.0 |                  5.0 | transform_p
| dropout_prob  | float   | 0.01       |     0.0 |     0.9 | 0.020556374027351852 | None
| lr_mult       | float   | 1.0        |     0.1 |    10.0 |    6.395423817618299 | None
| batch_size    | int     | 4          |     1.0 |     4.0 |                  4.0 | transform_p
| epochs        | int     | 4          |     2.0 |    16.0 |                 10.0 | transform_p
```

```
| k_folds      | int    | 1         |     1.0 |     1.0 |                  1.0 | None
| patience     | int    | 2         |     3.0 |     7.0 |                  5.0 | transform_p
| optimizer    | factor | SGD       |     0.0 |     6.0 |                  0.0 | None
| sgd_momentum | float  | 0.0       |     0.0 |     1.0 |   0.8840021468704181 | None
```

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

Figure 24.2: Variable importance plot, threshold 0.025.

24.10.1 Get the Tuned Architecture (SPOT Results)

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

```
Net_lin_reg(
  (fc1): Linear(in_features=10, out_features=32, bias=True)
  (fc2): Linear(in_features=32, out_features=16, bias=True)
  (fc3): Linear(in_features=16, out_features=1, bias=True)
  (relu): ReLU()
```

```
  (softmax): Softmax(dim=1)
  (dropout1): Dropout(p=0.020556374027351852, inplace=False)
  (dropout2): Dropout(p=0.010278187013675926, inplace=False)
)
```

24.10.2 Evaluation of the Tuned Architecture

```
from spotPython.torch.traintest import (
    train_tuned,
    test_tuned,
    )
```

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

```
Epoch: 1 | MeanAbsoluteError: 0.1087883487343788 | Loss: 0.0187988689071254 | Epoch: 2 | Mean

MeanAbsoluteError: 0.1999209225177765 | Loss: 0.0485347160383275 | Epoch: 6 | MeanAbsoluteEr

MeanAbsoluteError: 0.0468909963965416 | Loss: 0.0044010036110290 | Epoch: 20 | MeanAbsoluteE

MeanAbsoluteError: 0.0796051397919655 | Loss: 0.0100376617751623 | Epoch: 25 | MeanAbsoluteE

MeanAbsoluteError: 0.0460384450852871 | Loss: 0.0038444639159072 | Epoch: 40 | MeanAbsoluteE

MeanAbsoluteError: 0.0444340258836746 | Loss: 0.0032126457295626 | Epoch: 45 | MeanAbsoluteE
Returned to Spot: Validation loss: 0.0043042246742468135
```

If `path` is set to a filename, e.g., `path = "model_spot_trained.pt"`, the weights of the trained model will be loaded from this file.

```
test_tuned(net=model_spot, test_dataset=test,
            shuffle=False,
```

```
            loss_function=fun_control["loss_function"],
            metric=fun_control["metric_torch"],
            device = fun_control["device"],
            task=fun_control["task"],)
```

```
MeanAbsoluteError: 0.0552606433629990 | Loss: 0.0040031901444308 | Final evaluation: Validati
Final evaluation: Validation metric: 0.05526064336299896
----------------------------------------------
```

```
(0.004003190144430846, nan, tensor(0.0553))
```

24.10.3 Cross-validated Evaluations

- This is the evaluation that will be used in the comparison (evaluatecv has to be updated before, to get metric vlaues!):

```
from spotPython.torch.traintest import evaluate_cv
# modify k-kolds:
setattr(model_spot, "k_folds",  10)
df_eval, df_preds, df_metrics = evaluate_cv(net=model_spot,
            dataset=fun_control["data"],
            loss_function=fun_control["loss_function"],
            metric=fun_control["metric_torch"],
            task=fun_control["task"],
            writer=fun_control["writer"],
            writerId="model_spot_cv",
            device = fun_control["device"])
```

```
Fold: 1
Epoch: 1 | MeanAbsoluteError: 0.1464391201734543 | Loss: 0.0315676193152155 | Epoch: 2 | Mean

MeanAbsoluteError: 0.1293325871229172 | Loss: 0.0239712619887931 | Epoch: 4 | MeanAbsoluteEr

MeanAbsoluteError: 0.0998941138386726 | Loss: 0.0144623725542000 | Epoch: 12 | MeanAbsoluteE

MeanAbsoluteError: 0.0606867261230946 | Loss: 0.0053217368466513 | Epoch: 15 | MeanAbsoluteE

MeanAbsoluteError: 0.0479233898222446 | Loss: 0.0036649919514145 | Epoch: 23 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.0758475959300995 | Loss: 0.0072504186204502 | Epoch: 26 | MeanAbsoluteEr

MeanAbsoluteError: 0.0826917663216591 | Loss: 0.0077472019807569 | Epoch: 34 | MeanAbsoluteEr

MeanAbsoluteError: 0.0175925511866808 | Loss: 0.0012428153006892 | Epoch: 37 | MeanAbsoluteEr

Epoch: 45 | MeanAbsoluteError: 0.1219070926308632 | Loss: 0.0151595700798290 | Epoch: 46 | Me

Epoch: 48 | MeanAbsoluteError: 0.0188612584024668 | Loss: 0.0005349950848280 | Epoch: 49 | Me

MeanAbsoluteError: 0.0159396938979626 | Loss: 0.0004403380111658 | Epoch: 57 | MeanAbsoluteEr

MeanAbsoluteError: 0.0699406266212463 | Loss: 0.0050743183653269 | Epoch: 60 | MeanAbsoluteEr

Epoch: 68 | MeanAbsoluteError: 0.0428867489099503 | Loss: 0.0028713896338429 | Epoch: 69 | Me

MeanAbsoluteError: 0.0458002984523773 | Loss: 0.0035321696195751 | Epoch: 71 | MeanAbsoluteEr

MeanAbsoluteError: 0.0277180522680283 | Loss: 0.0015085020983991 | Epoch: 80 | MeanAbsoluteEr

MeanAbsoluteError: 0.0208641272038221 | Loss: 0.0006971032209029 | Epoch: 83 | MeanAbsoluteEr

MeanAbsoluteError: 0.0355649814009666 | Loss: 0.0020324253044756 | Epoch: 91 | MeanAbsoluteEr

MeanAbsoluteError: 0.0529253855347633 | Loss: 0.0036818779938455 | Epoch: 94 | MeanAbsoluteEr

Early stopping at epoch 100
Fold: 2
Epoch: 1 | MeanAbsoluteError: 0.1146809607744217 | Loss: 0.0199943981798632 | Epoch: 2 | Mean

MeanAbsoluteError: 0.0809583216905594 | Loss: 0.0119653018856687 | Epoch: 4 | MeanAbsoluteErr

MeanAbsoluteError: 0.0540966540575027 | Loss: 0.0048125730162220 | Epoch: 12 | MeanAbsoluteEr

Epoch: 15 | MeanAbsoluteError: 0.0510178506374359 | Loss: 0.0041383833823992 | Epoch: 16 | Me

MeanAbsoluteError: 0.0355682484805584 | Loss: 0.0019253144522996 | Epoch: 24 | MeanAbsoluteEr
```

```
MeanAbsoluteError: 0.0550338737666607 | Loss: 0.0037216772990567 | Epoch: 27 | MeanAbsoluteEr

MeanAbsoluteError: 0.0576522722840309 | Loss: 0.0045977485964873 | Epoch: 36 | MeanAbsoluteEr

MeanAbsoluteError: 0.0864447876811028 | Loss: 0.0083012187055179 | Epoch: 39 | MeanAbsoluteEr

MeanAbsoluteError: 0.0458258017897606 | Loss: 0.0023091279602210 | Epoch: 48 | MeanAbsoluteEr

MeanAbsoluteError: 0.0801960974931717 | Loss: 0.0066702176949808 | Early stopping at epoch 49
Fold: 3
Epoch: 1 | MeanAbsoluteError: 0.1035153493285179 | Loss: 0.0217744697417532 | Epoch: 2 | Mean

Epoch: 9 | MeanAbsoluteError: 0.0856635868549347 | Loss: 0.0101220122804599 | Epoch: 10 | Mea

Epoch: 12 | MeanAbsoluteError: 0.0413197726011276 | Loss: 0.0030982317236651 | Epoch: 13 | Me

MeanAbsoluteError: 0.0513975284993649 | Loss: 0.0052251401523660 | Epoch: 21 | MeanAbsoluteEr

MeanAbsoluteError: 0.0758574083447456 | Loss: 0.0060270660822945 | Epoch: 24 | MeanAbsoluteEr

MeanAbsoluteError: 0.0483733378350735 | Loss: 0.0037859969306737 | Epoch: 33 | MeanAbsoluteEr

MeanAbsoluteError: 0.0698173642158508 | Loss: 0.0064827473168927 | Epoch: 36 | MeanAbsoluteEr

MeanAbsoluteError: 0.0194466356188059 | Loss: 0.0010407603528750 | Epoch: 45 | MeanAbsoluteEr

MeanAbsoluteError: 0.0655636489391327 | Loss: 0.0047500019094774 | Epoch: 48 | MeanAbsoluteEr

MeanAbsoluteError: 0.0531153231859207 | Loss: 0.0031060285400599 | Epoch: 57 | MeanAbsoluteEr

MeanAbsoluteError: 0.0130973160266876 | Loss: 0.0004277561799557 | Epoch: 60 | MeanAbsoluteEr

MeanAbsoluteError: 0.0135004380717874 | Loss: 0.0003878879911749 | Epoch: 69 | MeanAbsoluteEr

MeanAbsoluteError: 0.0615096054971218 | Loss: 0.0044166241173765 | Epoch: 72 | MeanAbsoluteEr

MeanAbsoluteError: 0.0353872403502464 | Loss: 0.0018747070571408 | Epoch: 81 | MeanAbsoluteEr
```

```
MeanAbsoluteError: 0.1013695523142815 | Loss: 0.0106378183034914 | Epoch: 83 | MeanAbsoluteEr

MeanAbsoluteError: 0.0452271364629269 | Loss: 0.0023917141703090 | Epoch: 92 | MeanAbsoluteEr

MeanAbsoluteError: 0.0255935564637184 | Loss: 0.0009437226316160 | Epoch: 95 | MeanAbsoluteEr
Fold: 4
Epoch: 1 | MeanAbsoluteError: 0.2080202996730804 | Loss: 0.0586104680384908 | Epoch: 2 | Mean

Epoch: 5 | MeanAbsoluteError: 0.0836989283561707 | Loss: 0.0111846004479698 | Epoch: 6 | Mean

MeanAbsoluteError: 0.1205524057149887 | Loss: 0.0172012731699007 | Epoch: 8 | MeanAbsoluteErr

Epoch: 16 | MeanAbsoluteError: 0.0448844283819199 | Loss: 0.0031841434372057 | Epoch: 17 | Me

Epoch: 19 | MeanAbsoluteError: 0.0687187686562538 | Loss: 0.0063975305363004 | Epoch: 20 | Me

MeanAbsoluteError: 0.0619703717529774 | Loss: 0.0054489730724267 | Epoch: 28 | MeanAbsoluteEr

MeanAbsoluteError: 0.0373190604150295 | Loss: 0.0028206447771351 | Epoch: 31 | MeanAbsoluteEr

MeanAbsoluteError: 0.0686299353837967 | Loss: 0.0053421938791871 | Epoch: 39 | MeanAbsoluteEr

MeanAbsoluteError: 0.0215469673275948 | Loss: 0.0007076141399531 | Epoch: 42 | MeanAbsoluteEr

MeanAbsoluteError: 0.0511664189398289 | Loss: 0.0029278994438105 | Epoch: 50 | MeanAbsoluteEr

MeanAbsoluteError: 0.0727955102920532 | Loss: 0.0057182616687247 | Epoch: 54 | MeanAbsoluteEr

MeanAbsoluteError: 0.1113512888550758 | Loss: 0.0136328074815018 | Epoch: 62 | MeanAbsoluteEr

MeanAbsoluteError: 0.0472144931554794 | Loss: 0.0031249708429511 | Epoch: 66 | MeanAbsoluteEr

MeanAbsoluteError: 0.0307015031576157 | Loss: 0.0010897747457160 | Epoch: 74 | MeanAbsoluteEr

MeanAbsoluteError: 0.0622696392238140 | Loss: 0.0083712876907417 | Epoch: 78 | MeanAbsoluteEr

MeanAbsoluteError: 0.0408532433211803 | Loss: 0.0023664868702846 | Epoch: 86 | MeanAbsoluteEr
```

```
MeanAbsoluteError: 0.0225157253444195 | Loss: 0.0008249511426714 | Epoch: 90 | MeanAbsoluteEr

MeanAbsoluteError: 0.0133911203593016 | Loss: 0.0008468462827815 | Epoch: 98 | MeanAbsoluteEr

MeanAbsoluteError: 0.0419808179140091 | Loss: 0.0024582097207063 | Epoch: 102 | MeanAbsolute

MeanAbsoluteError: 0.0936533436179161 | Loss: 0.0093924580141902 | Epoch: 110 | MeanAbsolute

MeanAbsoluteError: 0.0791940763592720 | Loss: 0.0072315869453762 | Epoch: 114 | MeanAbsolute

MeanAbsoluteError: 0.0253671593964100 | Loss: 0.0009784159732849 | Epoch: 122 | MeanAbsolute

MeanAbsoluteError: 0.0314212404191494 | Loss: 0.0016765695763752 | Epoch: 126 | MeanAbsolute
Fold: 5
Epoch: 1 | MeanAbsoluteError: 0.1456587761640549 | Loss: 0.0294443110802344 | Epoch: 2 | Mean

MeanAbsoluteError: 0.1956333369016647 | Loss: 0.0474008625107152 | Epoch: 6 | MeanAbsoluteErr

MeanAbsoluteError: 0.1019653230905533 | Loss: 0.0138869997380035 | Epoch: 10 | MeanAbsoluteEr

MeanAbsoluteError: 0.0967235565185547 | Loss: 0.0122100269155843 | Epoch: 18 | MeanAbsoluteEr

MeanAbsoluteError: 0.0713557153940201 | Loss: 0.0075581834784576 | Epoch: 22 | MeanAbsoluteEr

MeanAbsoluteError: 0.1468367427587509 | Loss: 0.0222572282488857 | Epoch: 30 | MeanAbsoluteEr

MeanAbsoluteError: 0.1135496050119400 | Loss: 0.0137446965756161 | Epoch: 34 | MeanAbsoluteEr

MeanAbsoluteError: 0.0667129531502724 | Loss: 0.0051556930078992 | Epoch: 42 | MeanAbsoluteEr

MeanAbsoluteError: 0.0507564432919025 | Loss: 0.0031899401718485 | Epoch: 46 | MeanAbsoluteEr

MeanAbsoluteError: 0.0429333746433258 | Loss: 0.0030831804948061 | Epoch: 54 | MeanAbsoluteEr

MeanAbsoluteError: 0.0515586435794830 | Loss: 0.0031523973947125 | Epoch: 58 | MeanAbsoluteEr

MeanAbsoluteError: 0.0334962308406830 | Loss: 0.0020005155903553 | Epoch: 66 | MeanAbsoluteEr
```

```
MeanAbsoluteError: 0.0736852958798409 | Loss: 0.0062905504102153 | Epoch: 70 | MeanAbsoluteEr

MeanAbsoluteError: 0.0726227760314941 | Loss: 0.0058157944253513 | Epoch: 78 | MeanAbsoluteEr

MeanAbsoluteError: 0.0467705614864826 | Loss: 0.0025065582844296 | Epoch: 82 | MeanAbsoluteEr

MeanAbsoluteError: 0.0508484095335007 | Loss: 0.0044974998704025 | Epoch: 90 | MeanAbsoluteEr

MeanAbsoluteError: 0.0401734113693237 | Loss: 0.0019572763537456 | Epoch: 94 | MeanAbsoluteEr

MeanAbsoluteError: 0.0139307938516140 | Loss: 0.0004493865604413 | Epoch: 102 | MeanAbsoluteE

MeanAbsoluteError: 0.0418821573257446 | Loss: 0.0028236796081598 | Epoch: 106 | MeanAbsoluteE

MeanAbsoluteError: 0.0463141314685345 | Loss: 0.0034611012420750 | Epoch: 114 | MeanAbsoluteE

MeanAbsoluteError: 0.0618391856551170 | Loss: 0.0063531198538840 | Epoch: 118 | MeanAbsoluteE

MeanAbsoluteError: 0.0423794426023960 | Loss: 0.0022398588979351 | Epoch: 126 | MeanAbsoluteE
Fold: 6
Epoch: 1 | MeanAbsoluteError: 0.1634981483221054 | Loss: 0.0468927520726408 | Epoch: 2 |

MeanAbsoluteError: 0.1187281981110573 | Loss: 0.0204641954707248 | Epoch: 3 | MeanAbsoluteErr

MeanAbsoluteError: 0.0691978484392166 | Loss: 0.0079369333938562 | Epoch: 10 | MeanAbsoluteEr

MeanAbsoluteError: 0.0894083306193352 | Loss: 0.0098303804573204 | Epoch: 14 | MeanAbsoluteEr

MeanAbsoluteError: 0.1305016279220581 | Loss: 0.0178008076867887 | Epoch: 21 | MeanAbsoluteEr

MeanAbsoluteError: 0.0660159513354301 | Loss: 0.0078073956205377 | Epoch: 26 | MeanAbsoluteEr

MeanAbsoluteError: 0.0464369952678680 | Loss: 0.0027451533325283 | Epoch: 33 | MeanAbsoluteEr

MeanAbsoluteError: 0.0649165958166122 | Loss: 0.0072188362412687 | Epoch: 38 | MeanAbsoluteEr

MeanAbsoluteError: 0.0712300091981888 | Loss: 0.0065791459554540 | Epoch: 45 | MeanAbsoluteEr
```

```
MeanAbsoluteError: 0.0633815154433250 | Loss: 0.0049377898685634 | Epoch: 50 | MeanAbsoluteEr

MeanAbsoluteError: 0.0654458254575729 | Loss: 0.0062561939869608 | Epoch: 57 | MeanAbsoluteEr

MeanAbsoluteError: 0.0852290168404579 | Loss: 0.0079615701522146 | Epoch: 62 | MeanAbsoluteEr

MeanAbsoluteError: 0.0158761460334063 | Loss: 0.0005761952946029 | Epoch: 69 | MeanAbsoluteEr

MeanAbsoluteError: 0.0653296038508415 | Loss: 0.0072980205794530 | Epoch: 74 | MeanAbsoluteEr

MeanAbsoluteError: 0.0443416163325310 | Loss: 0.0022091833109568 | Epoch: 81 | MeanAbsoluteEr

MeanAbsoluteError: 0.0282034426927567 | Loss: 0.0013595724179010 | Epoch: 86 | MeanAbsoluteEr

MeanAbsoluteError: 0.0574185550212860 | Loss: 0.0035732639620879 | Epoch: 93 | MeanAbsoluteEr

MeanAbsoluteError: 0.0695087462663651 | Loss: 0.0050741509268326 | Epoch: 98 | MeanAbsoluteEr

MeanAbsoluteError: 0.0702792331576347 | Loss: 0.0051272570022515 | Epoch: 105 | MeanAbsoluteE

MeanAbsoluteError: 0.0562025792896748 | Loss: 0.0054952604363539 | Epoch: 110 | MeanAbsoluteE

MeanAbsoluteError: 0.0399301946163177 | Loss: 0.0025100902920323 | Epoch: 117 | MeanAbsoluteE

MeanAbsoluteError: 0.0562208443880081 | Loss: 0.0038918533828110 | Epoch: 122 | MeanAbsoluteE

MeanAbsoluteError: 0.0678830221295357 | Loss: 0.0053976918570697 | Epoch: 129 | MeanAbsoluteE

MeanAbsoluteError: 0.0277759358286858 | Loss: 0.0012439431863770 | Epoch: 134 | MeanAbsoluteE

MeanAbsoluteError: 0.0178364664316177 | Loss: 0.0005758742751953 | Epoch: 141 | MeanAbsoluteE

MeanAbsoluteError: 0.1037484332919121 | Loss: 0.0119391949847341 | Epoch: 146 | MeanAbsoluteE

MeanAbsoluteError: 0.0096889557316899 | Loss: 0.0002258220816397 | Epoch: 153 | MeanAbsoluteE

MeanAbsoluteError: 0.0675192177295685 | Loss: 0.0048487794452480 | Epoch: 157 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.0335980020463467 | Loss: 0.0015525644578572 | Epoch: 162 | MeanAbsoluteE

MeanAbsoluteError: 0.0563768856227398 | Loss: 0.0036012663040310 | Epoch: 169 | MeanAbsoluteE

MeanAbsoluteError: 0.0352589376270771 | Loss: 0.0013390039343254 | Epoch: 174 | MeanAbsoluteE
Fold: 7
Epoch: 1 | MeanAbsoluteError: 0.1155718043446541 | Loss: 0.0190127680876425 | Epoch: 2 | Mean

MeanAbsoluteError: 0.1041072830557823 | Loss: 0.0186116648837924 | Epoch: 6 | MeanAbsoluteErr

MeanAbsoluteError: 0.0528597012162209 | Loss: 0.0044480359314808 | Epoch: 11 | MeanAbsoluteEr

MeanAbsoluteError: 0.0517501123249531 | Loss: 0.0037145205839936 | Epoch: 18 | MeanAbsoluteEr

MeanAbsoluteError: 0.0790549442172050 | Loss: 0.0066287594049105 | Epoch: 23 | MeanAbsoluteEr

MeanAbsoluteError: 0.1015234962105751 | Loss: 0.0126085515533175 | Epoch: 30 | MeanAbsoluteEr

Epoch: 34 | MeanAbsoluteError: 0.0754304006695747 | Loss: 0.0060040986697589 | Epoch: 35 | Me

MeanAbsoluteError: 0.0937556847929955 | Loss: 0.0093327489282404 | Epoch: 42 | MeanAbsoluteEr

MeanAbsoluteError: 0.0357241556048393 | Loss: 0.0027404292320300 | Epoch: 46 | MeanAbsoluteEr

MeanAbsoluteError: 0.0231125876307487 | Loss: 0.0007425607265239 | Epoch: 54 | MeanAbsoluteEr

MeanAbsoluteError: 0.0505310446023941 | Loss: 0.0027025124789881 | Epoch: 58 | MeanAbsoluteEr

MeanAbsoluteError: 0.0530490651726723 | Loss: 0.0030514198421900 | Epoch: 66 | MeanAbsoluteEr

MeanAbsoluteError: 0.1328134387731552 | Loss: 0.0186151026615075 | Epoch: 70 | MeanAbsoluteEr

MeanAbsoluteError: 0.0186541657894850 | Loss: 0.0007113710785883 | Epoch: 78 | MeanAbsoluteEr

MeanAbsoluteError: 0.1210415139794350 | Loss: 0.0154564336740545 | Epoch: 82 | MeanAbsoluteEr

MeanAbsoluteError: 0.1088214889168739 | Loss: 0.0129703277988093 | Epoch: 90 | MeanAbsoluteEr
```

```
Epoch: 93 | MeanAbsoluteError: 0.0767060071229935 | Loss: 0.0062865880317986 | Epoch: 94 | Me

MeanAbsoluteError: 0.0374830067157745 | Loss: 0.0016989079969270 | Epoch: 102 | MeanAbsolutel

MeanAbsoluteError: 0.0339853204786777 | Loss: 0.0012804342134457 | Epoch: 105 | MeanAbsolutel

MeanAbsoluteError: 0.0638242959976196 | Loss: 0.0051982382511986 | Epoch: 113 | MeanAbsolutel

Epoch: 116 | MeanAbsoluteError: 0.0629240721464157 | Loss: 0.0041634018665978 | Epoch: 117 |
Fold: 8
Epoch: 1 | MeanAbsoluteError: 0.1819846332073212 | Loss: 0.0428598113358021 | Epoch: 2 | Mean

MeanAbsoluteError: 0.0815770328044891 | Loss: 0.0116155055750694 | Epoch: 4 | MeanAbsoluteErr

MeanAbsoluteError: 0.1182039901614189 | Loss: 0.0195558857438820 | Epoch: 7 | MeanAbsoluteErr

MeanAbsoluteError: 0.0665790811181068 | Loss: 0.0066286123889898 | Epoch: 15 | MeanAbsoluteEr

Epoch: 18 | MeanAbsoluteError: 0.0420612320303917 | Loss: 0.0024666726655726 | Epoch: 19 | Me

Epoch: 26 | MeanAbsoluteError: 0.0630166158080101 | Loss: 0.0044373538611191 | Epoch: 27 | Me

MeanAbsoluteError: 0.0636304989457130 | Loss: 0.0051788123923221 | Epoch: 30 | MeanAbsoluteEr

MeanAbsoluteError: 0.0266223922371864 | Loss: 0.0012436313305183 | Epoch: 38 | MeanAbsoluteEr

MeanAbsoluteError: 0.0625825375318527 | Loss: 0.0041846357219453 | Epoch: 42 | MeanAbsoluteEr

MeanAbsoluteError: 0.1120143234729767 | Loss: 0.0148500949144363 | Epoch: 50 | MeanAbsoluteEr

MeanAbsoluteError: 0.0177494138479233 | Loss: 0.0005236560952783 | Epoch: 54 | MeanAbsoluteEr

MeanAbsoluteError: 0.1131139099597931 | Loss: 0.0128631240555218 | Epoch: 62 | MeanAbsoluteEr

MeanAbsoluteError: 0.0795436426997185 | Loss: 0.0067660511205239 | Epoch: 66 | MeanAbsoluteEr

MeanAbsoluteError: 0.0517756380140781 | Loss: 0.0028496635412531 | Epoch: 73 | MeanAbsoluteEr
```

```
MeanAbsoluteError: 0.0766716375946999 | Loss: 0.0062038271155741 | Epoch: 77 | MeanAbsoluteEr

MeanAbsoluteError: 0.0718082934617996 | Loss: 0.0053434765764645 | Epoch: 84 | MeanAbsoluteEr

MeanAbsoluteError: 0.0767277181148529 | Loss: 0.0064341199052121 | Epoch: 88 | MeanAbsoluteEr

MeanAbsoluteError: 0.0643249154090881 | Loss: 0.0042474249910031 | Epoch: 96 | MeanAbsoluteEr

MeanAbsoluteError: 0.0705836713314056 | Loss: 0.0053772228503866 | Epoch: 100 | MeanAbsoluteE

Epoch: 107 | MeanAbsoluteError: 0.0503891371190548 | Loss: 0.0032896612371717 | Epoch: 108 |

Epoch: 111 | MeanAbsoluteError: 0.0717997997999191 | Loss: 0.0070373916200229 | Epoch: 112 |
Fold: 9
Epoch: 1 | MeanAbsoluteError: 0.2436308264732361 | Loss: 0.0676676056214741 | Epoch: 2 | Mean

Epoch: 4 | MeanAbsoluteError: 0.0641160011291504 | Loss: 0.0064381138820733 | Epoch: 5 | Mean

MeanAbsoluteError: 0.0507807508111000 | Loss: 0.0041665778096233 | Epoch: 9 | MeanAbsoluteErr

MeanAbsoluteError: 0.0871034264564514 | Loss: 0.0098408109375409 | Epoch: 16 | MeanAbsoluteEr

MeanAbsoluteError: 0.1040184721350670 | Loss: 0.0122232378593513 | Epoch: 21 | MeanAbsoluteEr

MeanAbsoluteError: 0.0301810745149851 | Loss: 0.0020930675340683 | Epoch: 28 | MeanAbsoluteEr

MeanAbsoluteError: 0.0197001975029707 | Loss: 0.0008856647182256 | Epoch: 33 | MeanAbsoluteEr

MeanAbsoluteError: 0.0494346320629120 | Loss: 0.0030724680816223 | Epoch: 40 | MeanAbsoluteEr

MeanAbsoluteError: 0.0401319824159145 | Loss: 0.0027793015054028 | Epoch: 45 | MeanAbsoluteEr

MeanAbsoluteError: 0.0828463807702065 | Loss: 0.0079741686183427 | Epoch: 52 | MeanAbsoluteEr

MeanAbsoluteError: 0.0433613397181034 | Loss: 0.0029331904037723 | Epoch: 57 | MeanAbsoluteEr
Fold: 10
Epoch: 1 |
```

```
MeanAbsoluteError: 0.1672105193138123 | Loss: 0.0386412808937686 | Epoch: 2 | MeanAbsoluteEr

MeanAbsoluteError: 0.1350432932376862 | Loss: 0.0245213168007987 | Epoch: 6 | MeanAbsoluteEr

MeanAbsoluteError: 0.1428214311599731 | Loss: 0.0256567538848945 | Epoch: 13 | MeanAbsoluteE

MeanAbsoluteError: 0.0612096078693867 | Loss: 0.0046038467303983 | Epoch: 18 | MeanAbsoluteE

MeanAbsoluteError: 0.0868644267320633 | Loss: 0.0081663414436792 | Epoch: 25 | MeanAbsoluteE

MeanAbsoluteError: 0.0682264044880867 | Loss: 0.0057603269815445 | Epoch: 30 | MeanAbsoluteE

MeanAbsoluteError: 0.0716345384716988 | Loss: 0.0055524547850447 | Epoch: 37 | MeanAbsoluteE

MeanAbsoluteError: 0.0390974618494511 | Loss: 0.0020865511947445 | Epoch: 42 | MeanAbsoluteE

MeanAbsoluteError: 0.0750031545758247 | Loss: 0.0071499954376902 | Epoch: 49 | MeanAbsoluteE

MeanAbsoluteError: 0.0629300475120544 | Loss: 0.0050624518854810 | Epoch: 54 | MeanAbsoluteE
```

```python
metric_name = type(fun_control["metric_torch"]).__name__
print(f"loss: {df_eval}, Cross-validated {metric_name}: {df_metrics}")
```

```
loss: 0.003777682612001497, Cross-validated MeanAbsoluteError: 0.052461277693510056
```

24.10.4 Detailed Hyperparameter Plots

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

```
l1:  12.777622230279281
dropout_prob:  0.8899440749165666
lr_mult:  48.517181667870446
epochs:  100.0
patience:  5.6484045873250786
sgd_momentum:  0.16434149645815124
```

Figure 24.3: Contour plots.

Surrogate
x4: epochs
x0: l1
0.021
0.019
0.017
0.015
0.013
0.011

0.020
0.018
0.016
0.014
0.012
x0: l1
x4: epochs

Surrogate
x5: patience
x0: l1
0.0175
0.0165
0.0155
0.0145
0.0135
0.0125
0.0115

0.016
0.014
0.012
x0: l1
x5: patience

Surrogate
x7: sgd_momentum
x0: l1
0.0174
0.0162
0.0150
0.0138
0.0126
0.0114

0.016
0.014
0.012
x0: l1
x7: sgd_momentum

Surrogate
x2: lr_mult
x1: dropout_prob
0.0180
0.0168
0.0156
0.0144
0.0132
0.0120
0.018
0.016
0.014
x1: dropout_prob
x2: lr_mult

Surrogate
x4: epochs
x1: dropout_prob
0.0200
0.0184
0.0168
0.0152
0.0136
0.0120
0.020
0.018
0.016
0.014
x1: dropout_prob
x4: epochs

Surrogate
x5: patience
x1: dropout_prob
0.0148
0.0144
0.0140
0.0136
0.0132
0.0128
0.0145
0.0140
0.0135
0.0130
x1: dropout_prob
x5: patience

Surrogate
x7: sgd_momentum
x1: dropout_prob
0.01395
0.01380
0.01365
0.01350
0.01335
0.01320
0.01305
0.01290
0.01275
0.01260
x1: dropout_prob
x7: sgd_momentum

Surrogate
x4: epochs
x2: lr_mult
0.0208
0.0192
0.0176
0.0160
0.0144
0.0128
x2: lr_mult
x4: epochs

Surrogate
x5: patience
x2: lr_mult
0.0175
0.0165
0.0155
0.0145
0.0135
0.0125
x2: lr_mult
x5: patience

Surrogate
x7: sgd_momentum
x2: lr_mult
0.0175
0.0165
0.0155
0.0145
0.0135
0.0125
x2: lr_mult
x7: sgd_momentum

Surrogate
x5: patience
x4: epochs
0.0208
0.0192
0.0176
0.0160
0.0144
0.0128
x4: epochs
x5: patience

Surrogate
x7: sgd_momentum
x4: epochs
0.0208
0.0192
0.0176
0.0160
0.0144
0.0128
x4: epochs
x7: sgd_momentum

24.10.5 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

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

Parallel coordinates plots

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

24.11 Summary and Outlook

This tutorial presents the hyperparameter tuning open source software `spotPython` for `PyTorch`. Some of the advantages of `spotPython` are:

- Numerical and categorical hyperparameters.
- Powerful surrogate models.
- Flexible approach and easy to use.
- Simple JSON files for the specification of the hyperparameters.
- Extension of default and user specified network classes.
- Noise handling techniques.
- Online visualization of the hyperparameter tuning process with `tensorboard`.

Currently, only rudimentary parallel and distributed neural network training is possible, but these capabilities will be extended in the future. The next version of `spotPython` will also include a more detailed documentation and more examples.

Important

Important: This tutorial does not present a complete benchmarking study (Bartz-Beielstein et al. 2020). The results are only preliminary and highly dependent on the local configuration (hard- and software). Our goal is to provide a first impression of the performance of the hyperparameter tuning package `spotPython`. The results should be interpreted with care.

25 HPT: PyTorch With VBDP

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

> Caution: Data must be downloaded manually
>
> - Ensure that the correspondiing data is available as `./data/VBDP/train.csv`.

This document refers to the following software versions:

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

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

```
spotPython                  0.2.38
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
```

25.1 Step 1: Setup

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

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

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

Note: Device selection

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

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

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

```
mps
```

```
import os
import copy
import socket
from datetime import datetime
from dateutil.tz import tzlocal
start_time = datetime.now(tzlocal())
HOSTNAME = socket.gethostname().split(".")[0]
experiment_name = '25-torch' + "_" + HOSTNAME + "_" + str(MAX_TIME) + "min_" + str(INIT_SI
experiment_name = experiment_name.replace(':', '-')
```

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

```
25-torch_bartz09_1min_5init_2023-06-19_01-05-34
```

25.2 Step 2: Initialization of the `fun_control` Dictionary

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

`spotPython` uses a Python dictionary for storing the information required for the hyperparameter tuning process, which was described in Section 19.2, see Initialization of the fun_control Dictionary in the documentation.

```
from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="classification",
    tensorboard_path="runs/25_spot_torch_vbdp",
    device=DEVICE)
```

25.3 Step 3: PyTorch Data Loading

25.3.1 1. Load VBDP Data

```
import pandas as pd
from sklearn.preprocessing import OrdinalEncoder
train_df = pd.read_csv('./data/VBDP/train.csv')
# remove the id column
train_df = train_df.drop(columns=['id'])
n_samples = train_df.shape[0]
n_features = train_df.shape[1] - 1
target_column = "prognosis"
# # 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.head()
```

```
# convert all entries to int for faster processing
train_df = train_df.astype(int)
```

- Add logical combinations (AND, OR, XOR) of the features to the data set:

```
from spotPython.utils.convert import add_logical_columns
df_new = train_df.copy()
# save the target column using "target_column" as the column name
target = train_df[target_column]
# remove the target column
df_new = df_new.drop(columns=[target_column])
train_df = add_logical_columns(df_new)
# add the target column back
train_df[target_column] = target
train_df = train_df.astype(int)
train_df.head()
```

	sudden_fever	headache	mouth_bleed	nose_bleed	muscle_pain	joint_pain	vomiting	rash	diar
0	1	1	0	1	1	1	1	0	1
1	0	0	0	0	0	0	1	0	1
2	0	1	1	1	0	1	1	1	1
3	0	0	1	1	1	1	0	1	0
4	0	0	0	0	0	0	0	0	1

```
from sklearn.model_selection import train_test_split
import numpy as np

n_samples = train_df.shape[0]
n_features = train_df.shape[1] - 1
train_df.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
train_df.head()
```

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	...	x6104	x6105	x6106	x6107	x6108	x6109	x61
0	1	1	0	1	1	1	1	0	1	1	...	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	0	1	0	...	0	0	0	0	0	0	0
2	0	1	1	1	0	1	1	1	1	1	...	1	1	0	1	1	0	1
3	0	0	1	1	1	1	0	1	0	1	...	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	1	0	...	0	1	1	0	1	1	0

25.3.2 Check content of the target column

```
train_df[target_column].head()
```

```
0     3
1     7
2     3
3    10
4     6
Name: prognosis, dtype: int64
```

```
X_train, X_test, y_train, y_test = train_test_split(train_df.drop(target_column, axis=1),
                                                    random_state=42,
                                                    test_size=0.25,
                                                    stratify=train_df[target_column])
trainset = pd.DataFrame(np.hstack((X_train, np.array(y_train).reshape(-1, 1))))
testset = pd.DataFrame(np.hstack((X_test, np.array(y_test).reshape(-1, 1))))
trainset.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
testset.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
print(train_df.shape)
print(trainset.shape)
print(testset.shape)
```

```
(707, 6113)
(530, 6113)
(177, 6113)
```

```
import torch
from sklearn.model_selection import train_test_split
from spotPython.torch.dataframedataset import DataFrameDataset
dtype_x = torch.float32
dtype_y = torch.long
train_df = DataFrameDataset(train_df, target_column=target_column, dtype_x=dtype_x, dtype_
train = DataFrameDataset(trainset, target_column=target_column, dtype_x=dtype_x, dtype_y=d
test = DataFrameDataset(testset, target_column=target_column, dtype_x=dtype_x, dtype_y=dty
n_samples = len(train)
```

```
# add the dataset to the fun_control
fun_control.update({"data": train_df, # full dataset,
                "train": train,
                "test": test,
                "n_samples": n_samples,
                "target_column": target_column})
```

25.4 Step 4: Specification of the Preprocessing Model

After the training and test data are specified and added to the `fun_control` dictionary, `spotPython` allows the specification of a data preprocessing pipeline, e.g., for the scaling of the data or for the one-hot encoding of categorical variables, see Section 19.4. This feature is not used here, so we do not change the default value (which is `None`).

25.5 Step 5: Select `algorithm` and `core_model_hyper_dict`

25.5.1 Implementing a Configurable Neural Network With spotPython

`spotPython` includes the `Net_vbdp` class which is implemented in the file `netvbdp.py`. The class is imported here.

This class inherits from the class `Net_Core` which is implemented in the file `netcore.py`, see Section 19.5.1.

25.5.2 Add the NN Model to the fun_control Dictionary

```
from spotPython.torch.netvbdp import Net_vbdp
from spotPython.data.torch_hyper_dict import TorchHyperDict
from spotPython.hyperparameters.values import add_core_model_to_fun_control
fun_control = add_core_model_to_fun_control(core_model=Net_vbdp,
                              fun_control=fun_control,
                              hyper_dict=TorchHyperDict)
```

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

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

```
{'_L0': {'type': 'int',
  'default': 64,
  'transform': 'None',
  'lower': 64,
  'upper': 64},
 'l1': {'type': 'int',
  'default': 8,
  'transform': 'transform_power_2_int',
  'lower': 8,
  'upper': 16},
 'dropout_prob': {'type': 'float',
  'default': 0.01,
  'transform': 'None',
  'lower': 0.0,
  'upper': 0.9},
 'lr_mult': {'type': 'float',
  'default': 1.0,
  'transform': 'None',
  'lower': 0.1,
  'upper': 10.0},
 'batch_size': {'type': 'int',
  'default': 4,
  'transform': 'transform_power_2_int',
  'lower': 1,
  'upper': 4},
 'epochs': {'type': 'int',
  'default': 4,
  'transform': 'transform_power_2_int',
  'lower': 4,
  'upper': 9},
 'k_folds': {'type': 'int',
  'default': 1,
  'transform': 'None',
  'lower': 1,
  'upper': 1},
 'patience': {'type': 'int',
  'default': 2,
  'transform': 'transform_power_2_int',
  'lower': 1,
  'upper': 5},
 'optimizer': {'levels': ['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}}
```

25.6 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 19.6.

Caution: Small number of epochs for demonstration purposes

- `epochs` and `patience` are set to small values for demonstration purposes. These values are too small for a real application.
- More resonable values are, e.g.:
  - `fun_control = modify_hyper_parameter_bounds(fun_control, "epochs", bounds=[7, 9])` and
  - `fun_control = modify_hyper_parameter_bounds(fun_control, "patience", bounds=[2, 7])`

```
from spotPython.hyperparameters.values import modify_hyper_parameter_bounds

fun_control = modify_hyper_parameter_bounds(fun_control, "_L0", bounds=[n_features, n_feat
fun_control = modify_hyper_parameter_bounds(fun_control, "l1", bounds=[6, 13])
fun_control = modify_hyper_parameter_bounds(fun_control, "epochs", bounds=[2, 3])
fun_control = modify_hyper_parameter_bounds(fun_control, "patience", bounds=[2, 2])
```

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

25.6.1 Optimizers

Optimizers are described in Section 19.6.1.

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

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

25.7.1 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 19.7.1)
2. the loss function (and a metric).

25.7.2 Loss Functions and Metrics

The loss function is specified by the key `"loss_function"`. We will use CrossEntropy loss for the multiclass-classification task.

```
from torch.nn import CrossEntropyLoss
loss_function = CrossEntropyLoss()
fun_control.update({"loss_function": loss_function})
```

25.7.3 Metric

- We will use the MAP@k metric for the evaluation of the model. Here is an example how this metric is calculated.

```
from spotPython.torch.mapk import MAPK
import torch
mapk = MAPK(k=2)
target = torch.tensor([0, 1, 2, 2])
preds = torch.tensor(
    [
        [0.5, 0.2, 0.2],  # 0 is in top 2
        [0.3, 0.4, 0.2],  # 1 is in top 2
        [0.2, 0.4, 0.3],  # 2 is in top 2
        [0.7, 0.2, 0.1],  # 2 isn't in top 2
    ]
)
mapk.update(preds, target)
print(mapk.compute()) # tensor(0.6250)
```

```
tensor(0.6250)
```

```
from spotPython.torch.mapk import MAPK
import torchmetrics
metric_torch = MAPK(k=3)
fun_control.update({"metric_torch": metric_torch})
```

25.8 Step 8: Calling the SPOT Function

25.8.1 Preparing the SPOT Call

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

```
# extract the variable types, names, and bounds
from spotPython.hyperparameters.values import (get_bound_values,
    get_var_name,
    get_var_type,)
var_type = get_var_type(fun_control)
var_name = get_var_name(fun_control)
```

```
fun_control.update({"var_type": var_type,
                    "var_name": var_name})
lower = get_bound_values(fun_control, "lower")
upper = get_bound_values(fun_control, "upper")
```

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

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

```
| name         | type   | default   |    lower |    upper | transform             |
|--------------|--------|-----------|----------|----------|-----------------------|
| _L0          | int    | 64        | 6112     | 6112     | None                  |
| l1           | int    | 8         |    6     |   13     | transform_power_2_int |
| dropout_prob | float  | 0.01      |    0     |    0.9   | None                  |
| lr_mult      | float  | 1.0       |    0.001 |    0.001 | None                  |
| batch_size   | int    | 4         |    1     |    4     | transform_power_2_int |
| epochs       | int    | 4         |    2     |    3     | transform_power_2_int |
| k_folds      | int    | 1         |    1     |    1     | None                  |
| patience     | int    | 2         |    2     |    2     | transform_power_2_int |
| optimizer    | factor | SGD       |    0     |    3     | None                  |
| sgd_momentum | float  | 0.0       |    0.9   |    0.9   | None                  |
```

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

25.8.2 The Objective Function `fun_torch`

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

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

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

25.8.3 Starting the Hyperparameter Tuning

The `spotPython` hyperparameter tuning is started by calling the `Spot` function as described in Section 19.8.4.

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

```
config: {'_L0': 6112, 'l1': 2048, 'dropout_prob': 0.17031221661559992, 'lr_mult': 0.001, 'ba
Epoch: 1 |

MAPK: 0.1279761940240860 | Loss: 2.3981505462101529 | Acc: 0.0518867924528302.
```

```
Epoch: 2 |

MAPK: 0.1302083432674408 | Loss: 2.3981248480933055 | Acc: 0.0518867924528302.
Epoch: 3 |

MAPK: 0.1302083283662796 | Loss: 2.3981098788125172 | Acc: 0.0518867924528302.
Epoch: 4 |

MAPK: 0.1264881044626236 | Loss: 2.3980372633252824 | Acc: 0.0518867924528302.
Epoch: 5 |

MAPK: 0.1331845223903656 | Loss: 2.3980248144694736 | Acc: 0.0518867924528302.
Epoch: 6 |

MAPK: 0.1316964328289032 | Loss: 2.3979563542774747 | Acc: 0.0518867924528302.
Epoch: 7 |

MAPK: 0.1346726119518280 | Loss: 2.3979602030345371 | Acc: 0.0518867924528302.
Epoch: 8 |

MAPK: 0.1369047611951828 | Loss: 2.3978721243994578 | Acc: 0.0518867924528302.
Returned to Spot: Validation loss: 2.397872124399458

config: {'_L0': 6112, 'l1': 256, 'dropout_prob': 0.19379790035512987, 'lr_mult': 0.001, 'bat
Epoch: 1 |

MAPK: 0.1612654328346252 | Loss: 2.3983733124203153 | Acc: 0.0943396226415094.
Epoch: 2 |

MAPK: 0.1612654328346252 | Loss: 2.3982823689778647 | Acc: 0.0943396226415094.
Epoch: 3 |

MAPK: 0.1612654328346252 | Loss: 2.3982720728273743 | Acc: 0.0943396226415094.
Epoch: 4 |

MAPK: 0.1612654328346252 | Loss: 2.3982368840111628 | Acc: 0.0943396226415094.
Returned to Spot: Validation loss: 2.3982368840111628

config: {'_L0': 6112, 'l1': 4096, 'dropout_prob': 0.6759063718076167, 'lr_mult': 0.001, 'bat
Epoch: 1 |
```

```
MAPK: 0.2287735491991043 | Loss: 2.3975862647002599 | Acc: 0.1603773584905660.
Epoch: 2 |

MAPK: 0.2209119498729706 | Loss: 2.3974618079527370 | Acc: 0.1509433962264151.
Epoch: 3 |

MAPK: 0.2114779651165009 | Loss: 2.3973085947756498 | Acc: 0.1273584905660377.
Epoch: 4 |

MAPK: 0.2264150828123093 | Loss: 2.3970519619167976 | Acc: 0.1415094339622641.
Epoch: 5 |

MAPK: 0.2405660152435303 | Loss: 2.3967285336188549 | Acc: 0.1509433962264151.
Epoch: 6 |

MAPK: 0.2248427569866180 | Loss: 2.3962760758849808 | Acc: 0.1273584905660377.
Epoch: 7 |

MAPK: 0.2138364613056183 | Loss: 2.3956628048195028 | Acc: 0.1179245283018868.
Epoch: 8 |

MAPK: 0.2051886767148972 | Loss: 2.3949478369838788 | Acc: 0.1132075471698113.
Returned to Spot: Validation loss: 2.394947836983879

config: {'_L0': 6112, 'l1': 128, 'dropout_prob': 0.37306669346546995, 'lr_mult': 0.001, 'bat
Epoch: 1 |

MAPK: 0.2193396389484406 | Loss: 2.3966948491222455 | Acc: 0.1556603773584906.
Epoch: 2 |

MAPK: 0.2193396389484406 | Loss: 2.3966967474739507 | Acc: 0.1556603773584906.
Epoch: 3 |

MAPK: 0.2201257944107056 | Loss: 2.3968763036547966 | Acc: 0.1556603773584906.
Epoch: 4 |

MAPK: 0.2193396240472794 | Loss: 2.3967286730712316 | Acc: 0.1556603773584906.
Returned to Spot: Validation loss: 2.3967286730712316

config: {'_L0': 6112, 'l1': 1024, 'dropout_prob': 0.870137281216666, 'lr_mult': 0.001, 'batc
Epoch: 1 |
```

```
MAPK: 0.1851851791143417 | Loss: 2.3978676089534052 | Acc: 0.1084905660377359.
Epoch: 2 |

MAPK: 0.1635802537202835 | Loss: 2.3979951893841780 | Acc: 0.0896226415094340.
Epoch: 3 |

MAPK: 0.1836419850587845 | Loss: 2.3978228392424406 | Acc: 0.0990566037735849.
Epoch: 4 |

MAPK: 0.1782407611608505 | Loss: 2.3978858347292298 | Acc: 0.1132075471698113.
Epoch: 5 |

MAPK: 0.1628086715936661 | Loss: 2.3976472307134560 | Acc: 0.0754716981132075.
Epoch: 6 |

MAPK: 0.1651234626770020 | Loss: 2.3977509516256825 | Acc: 0.0849056603773585.
Epoch: 7 |

MAPK: 0.1766975373029709 | Loss: 2.3977770452146179 | Acc: 0.0896226415094340.
Epoch: 8 |

MAPK: 0.1905864030122757 | Loss: 2.3977227917423956 | Acc: 0.1179245283018868.
Returned to Spot: Validation loss: 2.3977227917423956

config: {'_L0': 6112, 'l1': 8192, 'dropout_prob': 0.7137757463039506, 'lr_mult': 0.001, 'bat

Epoch: 1 |

MAPK: 0.1800314337015152 | Loss: 2.3975703131477788 | Acc: 0.0896226415094340.
Epoch: 2 |

MAPK: 0.1933962106704712 | Loss: 2.3974594575054242 | Acc: 0.0943396226415094.
Epoch: 3 |

MAPK: 0.2051886916160583 | Loss: 2.3970677312814965 | Acc: 0.1084905660377359.
Epoch: 4 |

MAPK: 0.2130503058433533 | Loss: 2.3964644670486450 | Acc: 0.1084905660377359.
Epoch: 5 |
```

```
MAPK: 0.2083333432674408 | Loss: 2.3955366184126654 | Acc: 0.1084905660377359.
Epoch: 6 |

MAPK: 0.2122641354799271 | Loss: 2.3944760853389524 | Acc: 0.1084905660377359.
Epoch: 7 |

MAPK: 0.1933962404727936 | Loss: 2.3933550861646546 | Acc: 0.1084905660377359.
Epoch: 8 |

MAPK: 0.1839622706174850 | Loss: 2.3919659628058381 | Acc: 0.1084905660377359.
Returned to Spot: Validation loss: 2.391965962805838
spotPython tuning: 2.391965962805838 [#######---] 73.04%


config: {'_L0': 6112, 'l1': 512, 'dropout_prob': 0.716041757199382, 'lr_mult': 0.001, 'batch_
Epoch: 1 |

MAPK: 0.1592261940240860 | Loss: 2.3980996779033115 | Acc: 0.0707547169811321.
Epoch: 2 |

MAPK: 0.1718750000000000 | Loss: 2.3979180199759349 | Acc: 0.0896226415094340.
Epoch: 3 |

MAPK: 0.1793154776096344 | Loss: 2.3979111228670393 | Acc: 0.1084905660377359.
Epoch: 4 |

MAPK: 0.1733630895614624 | Loss: 2.3979867526463101 | Acc: 0.0896226415094340.
Returned to Spot: Validation loss: 2.39798675264631
spotPython tuning: 2.391965962805838 [#######---] 74.83%


config: {'_L0': 6112, 'l1': 8192, 'dropout_prob': 0.856772479725372, 'lr_mult': 0.001, 'batch

Epoch: 1 |

MAPK: 0.1981132328510284 | Loss: 2.3975356974691713 | Acc: 0.1226415094339623.
Epoch: 2 |
```

```
MAPK: 0.2051886767148972 | Loss: 2.3971331142029673 | Acc: 0.1226415094339623.
Epoch: 3 |

MAPK: 0.2295597344636917 | Loss: 2.3966757306512796 | Acc: 0.1179245283018868.
Epoch: 4 |

MAPK: 0.2319182455539703 | Loss: 2.3960931705978683 | Acc: 0.1179245283018868.
Epoch: 5 |

MAPK: 0.2209119498729706 | Loss: 2.3958084425836241 | Acc: 0.1226415094339623.
Epoch: 6 |

MAPK: 0.2106918245553970 | Loss: 2.3951764219212084 | Acc: 0.1132075471698113.
Epoch: 7 |

MAPK: 0.2130503207445145 | Loss: 2.3943770988932194 | Acc: 0.1179245283018868.
Epoch: 8 |

MAPK: 0.2099056392908096 | Loss: 2.3940468891611637 | Acc: 0.1179245283018868.
Returned to Spot: Validation loss: 2.3940468891611637
spotPython tuning: 2.391965962805838 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x16e08b220>
```

25.9 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section 19.9, see also the description in the documentation: Tensorboard.

25.10 Step 10: Results

After the hyperparameter tuning run is finished, the results can be analyzed as described in Section 19.10.

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

Figure 25.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

```
from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control=fun_control, spot=spot_tuner))
```

```
| name          | type    | default    |   lower |   upper |               tuned | transform
|---------------|---------|------------|---------|---------|---------------------|---------------
| _L0           | int     | 64         |  6112.0 |  6112.0 |              6112.0 | None
| l1            | int     | 8          |     6.0 |    13.0 |                13.0 | transform_powe
| dropout_prob  | float   | 0.01       |     0.0 |     0.9 | 0.7137757463039506 | None
| lr_mult       | float   | 1.0        |   0.001 |   0.001 |               0.001 | None
| batch_size    | int     | 4          |     1.0 |     4.0 |                 1.0 | transform_powe
| epochs        | int     | 4          |     2.0 |     3.0 |                 3.0 | transform_powe
| k_folds       | int     | 1          |     1.0 |     1.0 |                 1.0 | None
| patience      | int     | 2          |     2.0 |     2.0 |                 2.0 | transform_powe
| optimizer     | factor  | SGD        |     0.0 |     3.0 |                 3.0 | None
| sgd_momentum  | float   | 0.0        |     0.9 |     0.9 |                 0.9 | None
```

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

Figure 25.2: Variable importance plot, threshold 0.025.

25.10.1 Get the Tuned Architecture

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

```
Net_vbdp(
  (fc1): Linear(in_features=6112, out_features=8192, bias=True)
  (fc2): Linear(in_features=8192, out_features=4096, bias=True)
  (fc3): Linear(in_features=4096, out_features=2048, bias=True)
  (fc4): Linear(in_features=2048, out_features=1024, bias=True)
  (fc5): Linear(in_features=1024, out_features=11, bias=True)
  (relu): ReLU()
  (softmax): Softmax(dim=1)
  (dropout1): Dropout(p=0.7137757463039506, inplace=False)
  (dropout2): Dropout(p=0.3568878731519753, inplace=False)
)
```

25.10.2 Evaluation of the Tuned Architecture

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

```
Epoch: 1 |

MAPK: 0.1902515888214111 | Loss: 2.3976921135524534 | Acc: 0.1037735849056604.
Epoch: 2 |

MAPK: 0.2004716992378235 | Loss: 2.3972881852455861 | Acc: 0.1179245283018868.
Epoch: 3 |

MAPK: 0.2130502909421921 | Loss: 2.3968124929464087 | Acc: 0.1179245283018868.
Epoch: 4 |

MAPK: 0.2264150828123093 | Loss: 2.3960668096002542 | Acc: 0.1179245283018868.
Epoch: 5 |

MAPK: 0.2146226167678833 | Loss: 2.3953527234635263 | Acc: 0.1179245283018868.
Epoch: 6 |

MAPK: 0.2146226167678833 | Loss: 2.3944759953696773 | Acc: 0.1179245283018868.
Epoch: 7 |

MAPK: 0.2122641503810883 | Loss: 2.3928036082465693 | Acc: 0.1179245283018868.
Epoch: 8 |

MAPK: 0.2114779651165009 | Loss: 2.3909273912321845 | Acc: 0.1179245283018868.
Returned to Spot: Validation loss: 2.3909273912321845
```

If `path` is set to a filename, e.g., `path = "model_spot_trained.pt"`, the weights of the trained model will be loaded from this file.

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

```
MAPK: 0.2181648015975952 | Loss: 2.3907887319500527 | Acc: 0.1186440677966102.
Final evaluation: Validation loss: 2.3907887319500527
Final evaluation: Validation metric: 0.21816480159759521
----------------------------------------------
```

```
(2.3907887319500527, nan, tensor(0.2182))
```

25.10.3 Cross-validated Evaluations

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

> Caution: Cross-validated Evaluations
>
> - The number of folds is set to 1 by default.
> - Here it was changed to 3 for demonstration purposes.
> - Set the number of folds to a reasonable value, e.g., 10.
> - This can be done by setting the `k_folds` attribute of the model as follows:
> - `setattr(model_spot, "k_folds",  10)`

```
from spotPython.torch.traintest import evaluate_cv
# modify k-kolds:
setattr(model_spot, "k_folds",  3)
df_eval, df_preds, df_metrics = evaluate_cv(net=model_spot,
    dataset=fun_control["data"],
    loss_function=fun_control["loss_function"],
    metric=fun_control["metric_torch"],
    task=fun_control["task"],
    writer=fun_control["writer"],
    writerId="model_spot_cv",
    device = fun_control["device"])
```

```
Fold: 1
Epoch: 1 |

MAPK: 0.2097457200288773 | Loss: 2.3973414635254167 | Acc: 0.1186440677966102.
Epoch: 2 |

MAPK: 0.2372880876064301 | Loss: 2.3965904167142966 | Acc: 0.1271186440677966.
Epoch: 3 |

MAPK: 0.2168079018592834 | Loss: 2.3956296888448425 | Acc: 0.1271186440677966.
Epoch: 4 |

MAPK: 0.2182203084230423 | Loss: 2.3942999536708252 | Acc: 0.1271186440677966.
Epoch: 5 |

MAPK: 0.2210451662540436 | Loss: 2.3915213225251537 | Acc: 0.1271186440677966.
Epoch: 6 |

MAPK: 0.2217513918876648 | Loss: 2.3886974666078213 | Acc: 0.1271186440677966.
Epoch: 7 |

MAPK: 0.2210451662540436 | Loss: 2.3862525426735313 | Acc: 0.1271186440677966.
Epoch: 8 |

MAPK: 0.2203389406204224 | Loss: 2.3840851985802085 | Acc: 0.1271186440677966.
Fold: 2
Epoch: 1 |

MAPK: 0.1892655491828918 | Loss: 2.3973272089230813 | Acc: 0.0974576271186441.
Epoch: 2 |

MAPK: 0.1942090392112732 | Loss: 2.3965696379289789 | Acc: 0.1186440677966102.
Epoch: 3 |

MAPK: 0.1899717599153519 | Loss: 2.3956974785206682 | Acc: 0.1186440677966102.
Epoch: 4 |

MAPK: 0.2104519754648209 | Loss: 2.3935168880527304 | Acc: 0.1186440677966102.
Epoch: 5 |
```

```
MAPK: 0.2083333283662796 | Loss: 2.3906324051194270 | Acc: 0.1186440677966102.
Epoch: 6 |

MAPK: 0.2125706076622009 | Loss: 2.3880335678488520 | Acc: 0.1186440677966102.
Epoch: 7 |

MAPK: 0.2288135290145874 | Loss: 2.3864956791118042 | Acc: 0.1186440677966102.
Epoch: 8 |

MAPK: 0.2620056271553040 | Loss: 2.3838293996907898 | Acc: 0.1567796610169492.
Fold: 3
Epoch: 1 |

MAPK: 0.2040960192680359 | Loss: 2.3972686812029047 | Acc: 0.1234042553191489.
Epoch: 2 |

MAPK: 0.2026836127042770 | Loss: 2.3964144456184515 | Acc: 0.1148936170212766.
Epoch: 3 |

MAPK: 0.2026835680007935 | Loss: 2.3952106378846247 | Acc: 0.1148936170212766.
Epoch: 4 |

MAPK: 0.1906779706478119 | Loss: 2.3932935706639693 | Acc: 0.1148936170212766.
Epoch: 5 |

MAPK: 0.1878530979156494 | Loss: 2.3905958119085278 | Acc: 0.1148936170212766.
Epoch: 6 |

MAPK: 0.1878531128168106 | Loss: 2.3878873586654663 | Acc: 0.1148936170212766.
Epoch: 7 |

MAPK: 0.1892655193805695 | Loss: 2.3861786329140098 | Acc: 0.1148936170212766.
Epoch: 8 |

MAPK: 0.1942090392112732 | Loss: 2.3843317476369568 | Acc: 0.1148936170212766.
```

```python
metric_name = type(fun_control["metric_torch"]).__name__
print(f"loss: {df_eval}, Cross-validated {metric_name}: {df_metrics}")
```

```
loss: 2.3840821153026517, Cross-validated MAPK: 0.2255178689956665
```

25.10.4 Detailed Hyperparameter Plots

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

```
l1:  100.0
batch_size:  30.01801303336478
optimizer:  46.884536789954986
```

Figure 25.3: Contour plots.

25.10.5 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

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

Parallel coordinates plots

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

```
# close tensorbaoard writer
if fun_control["writer"] is not None:
    fun_control["writer"].close()
```

25.10.6 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

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

26 Documentation of the Sequential Parameter Optimization

This document describes the `Spot` features.

26.1 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
```

26.1.1 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`.

26.1.2 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`

26.2 The `fun_control` Dictionary

external parameter	type	description	default	mandatory
`sigma`	float	noise: standard deviation	`0`	yes
`seed`	int	seed for rng	`124`	yes

26.3 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

26.4 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`	

26.5 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

26.6 Run

```
spot_1.run()
```

```
<spotPython.spot.spot.Spot at 0x17ea869e0>
```

26.7 Print the Results

```
spot_1.print_results()
```

```
min y: 0.30656551286610595
x0: -0.5536835855126157

[['x0', -0.5536835855126157]]
```

26.8 Show the Progress

```
spot_1.plot_progress()
```

26.9 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>
```

26.10 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]
```

26.11 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 ]]))
```

26.12 Surrogates

26.12.1 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.

26.13 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  0.31802454 -0.68898927 -0.75129705  0.97550354
  0.41757584  0.0786237   0.82585329  0.23700598 -0.49274073 -0.82319082
 -0.17991251  0.1481835 ]

[nan]

[-1.]

[-0.58552368]
[-0.20126111]

[-0.60100809]
[-0.97897336]

[-0.2748985]
[0.8359486]

[0.99035591]
[0.01641232]

[0.5629346]
```

26.14 PyTorch: Detailed Description of the Data Splitting

26.14.1 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
```

26.14.1.1 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.

26.14.1.2 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`.

26.14.1.3 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.

26.14.1.4 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.