Hyperparameter Tuning Cookbook

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

Thomas Bartz-Beielstein

Jun 27, 2023

Table of contents

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

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

Section 14.11 presents a summary and an outlook.

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

1.1 The Hyperparameter Tuning Software SPOT

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

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

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

1. Loading the data (training and test datasets), see Section 14.3.
2. Specification of the preprocessing model, see Section 14.4. This model is called `prep_model` ("preparation" or pre-processing). The information required for the hyperparameter tuning is stored in the dictionary `fun_control`. Thus, the information needed for the execution of the hyperparameter tuning is available in a readable form.
3. Selection of the machine learning or deep learning model to be tuned, see Section 14.5. This is called the `core_model`. Once the `core_model` is defined, then the associated hyperparameters are stored in the `fun_control` dictionary. First, the hyperparameters of the `core_model` are initialized with the default values of the `core_model`. As default values we use the default values contained in the `spotPython` package for the algorithms of the `torch` package.
4. Modification of the default values for the hyperparameters used in `core_model`, see Section 14.6.0.1. This step is optional.

    1. numeric parameters are modified by changing the bounds.
    2. categorical parameters are modified by changing the categories ("levels").

5. Selection of target function (loss function) for the optimizer, see Section 14.7.5.
6. Calling SPOT with the corresponding parameters, see Section 14.8.4. The results are stored in a dictionary and are available for further analysis.
7. Presentation, visualization and interpretation of the results, see Section 14.10.

1.2 Spot as an Optimizer

The `spot` loop consists of the following steps:

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

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

1.3 Example: `Spot` and the Sphere Function

```python
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

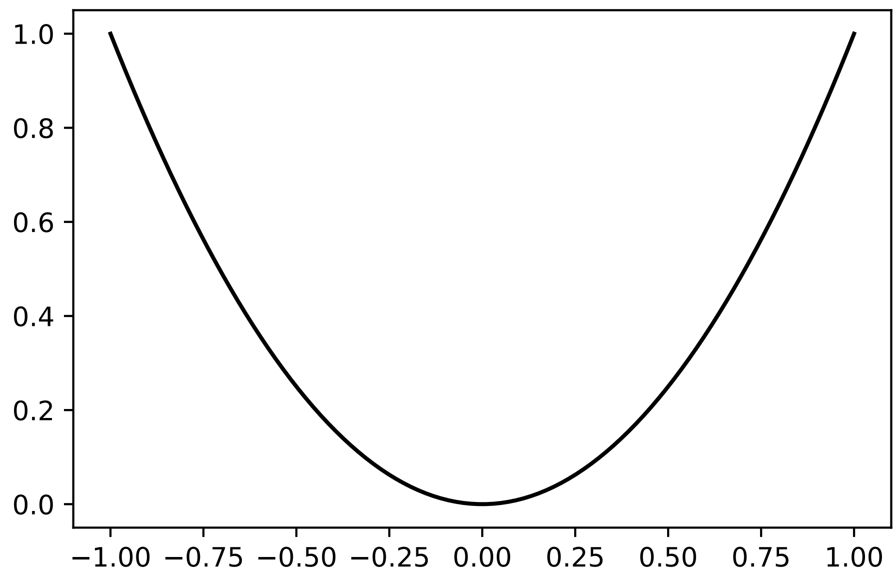
The `spotPython` package provides several classes of objective functions. We will use an analytical objective function, i.e., a function that can be described by a (closed) formula:

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

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

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

```python
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()
```

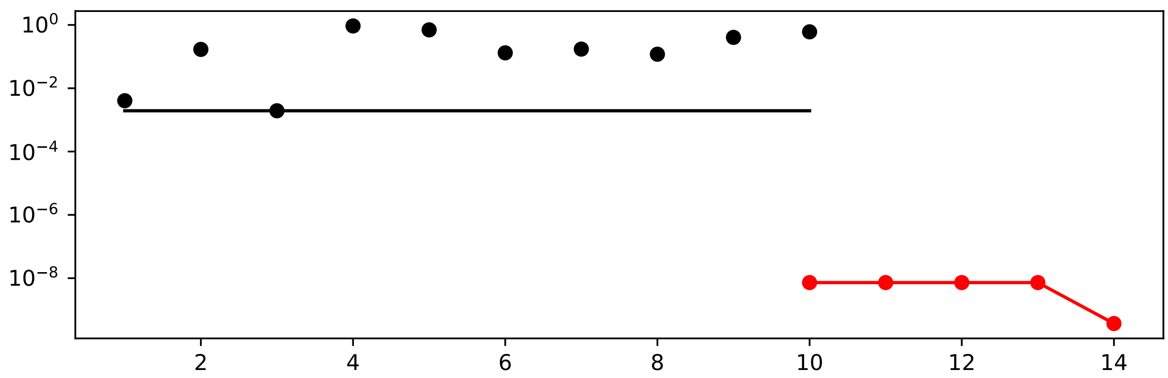
```
<spotPython.spot.spot.Spot at 0x16ce28bb0>
```

```
spot_0.print_results()
```

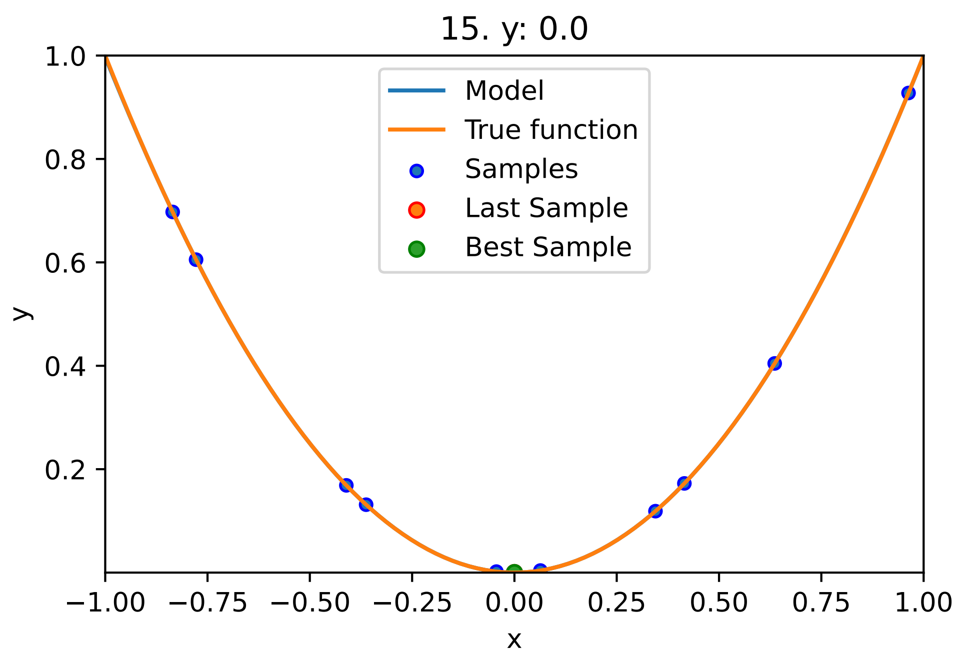
```
min y: 3.696886711914087e-10
x0: 1.922728975158508e-05
```

```
[['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`

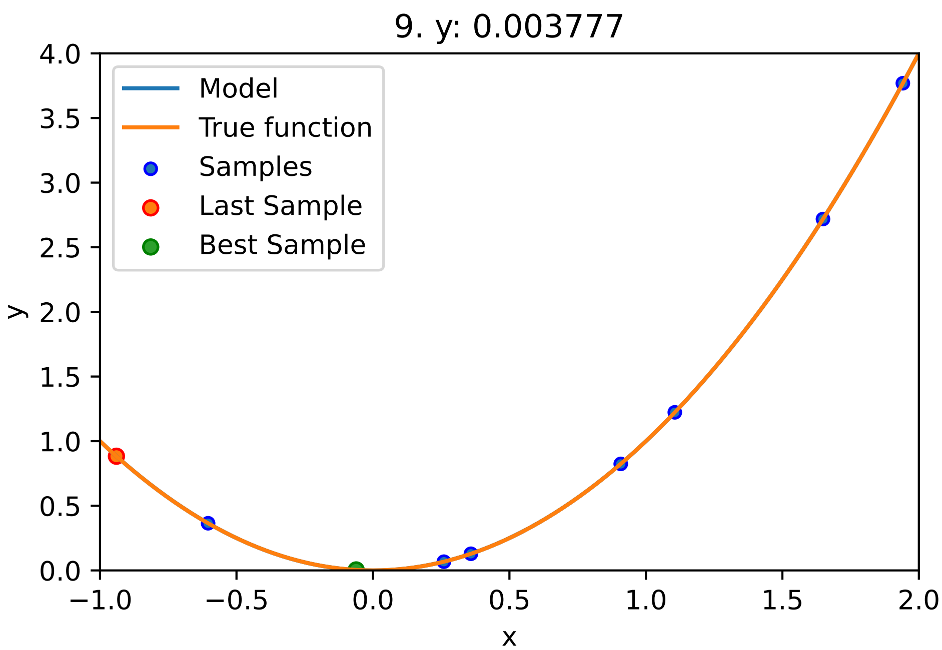
We will modify three parameters:

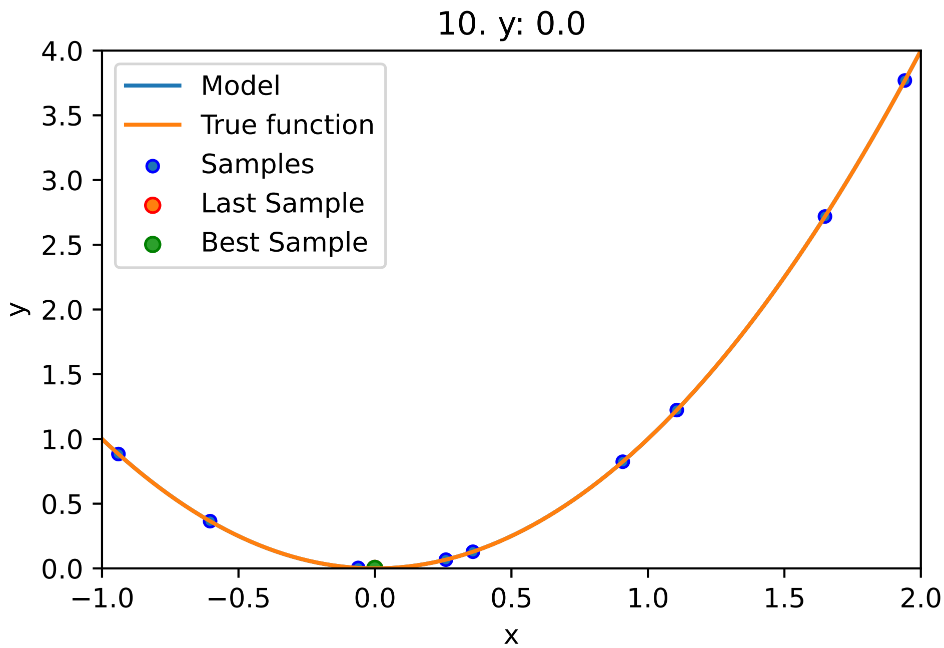
1. The number of function evaluations (`fun_evals`)
2. The size of the initial design (`init_size`)

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 0x16d6cfe50>
```

1.5 Print the Results

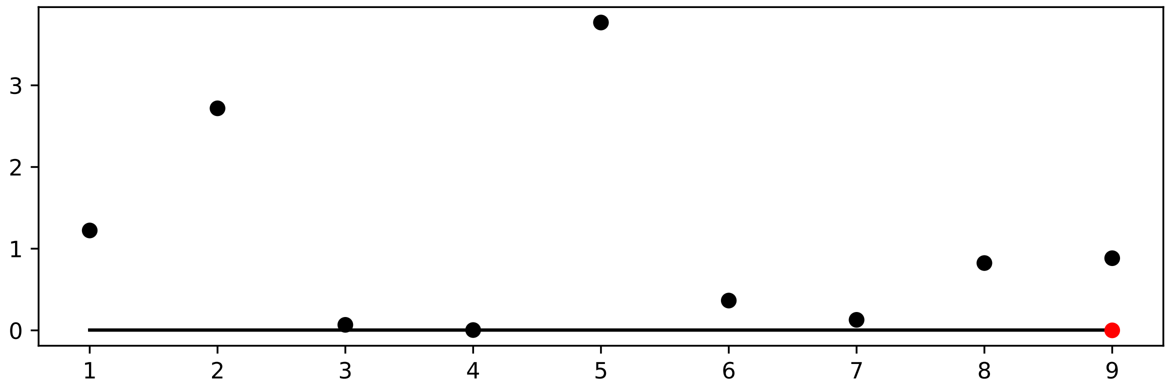
```
spot_1.print_results()
```

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

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

1.6 Show the Progress

```
spot_1.plot_progress()
```


0
1
2
3
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()
```

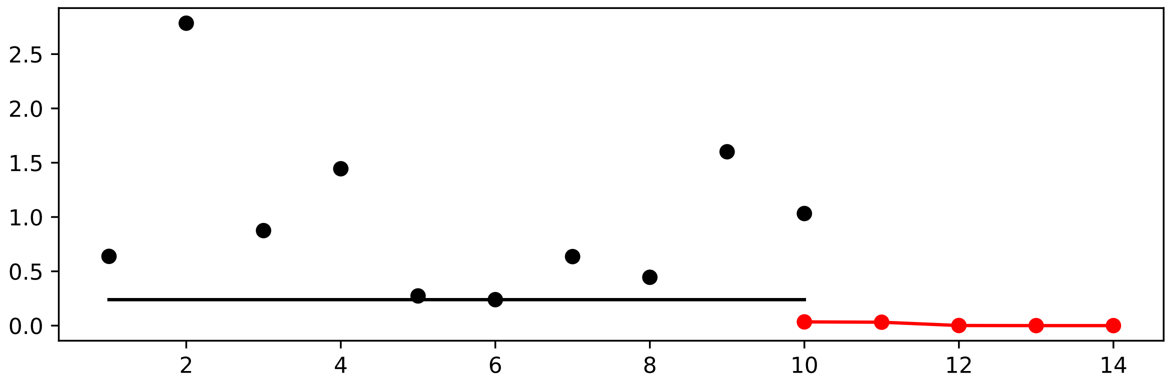
```
spotPython tuning: 0.03443344056467332 [#######---] 73.33%

spotPython tuning: 0.031348659935079 26 [########--] 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 0x109336b30>
```

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]]
```

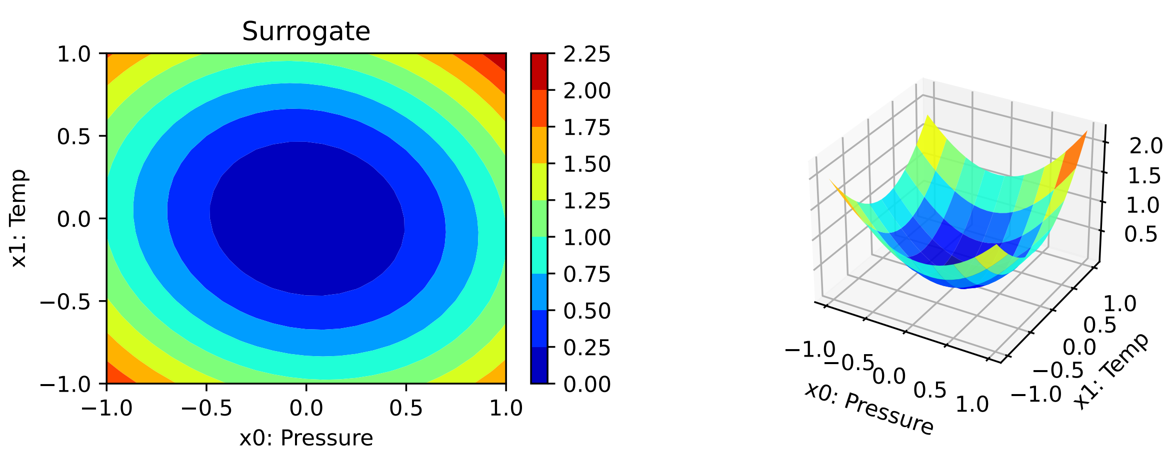
```
spot_3.plot_progress()
```



2.1.3 A Contour Plot

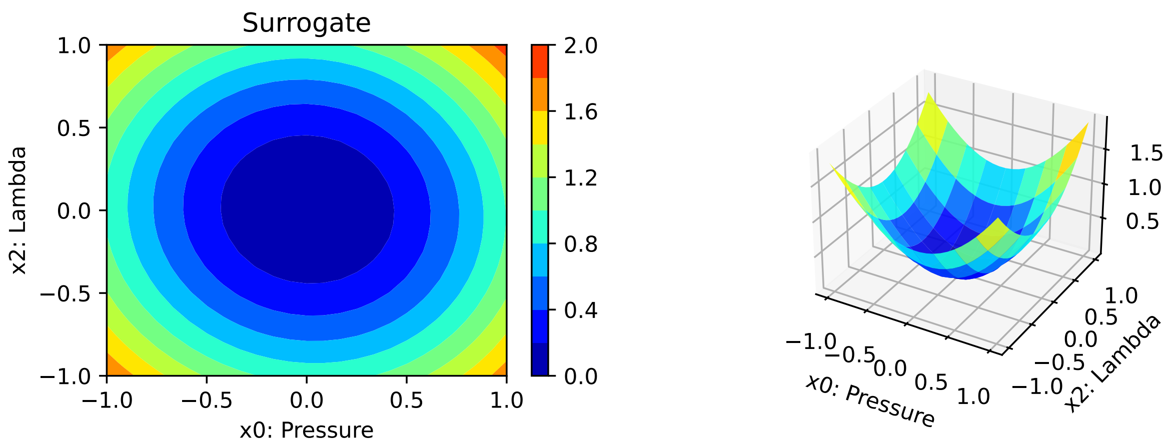
- We can select two dimensions, say $i = 0$ and $j = 1$, and generate a contour plot as follows.
  - Note: We have specified identical `min_z` and `max_z` values to generate comparable plots!

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



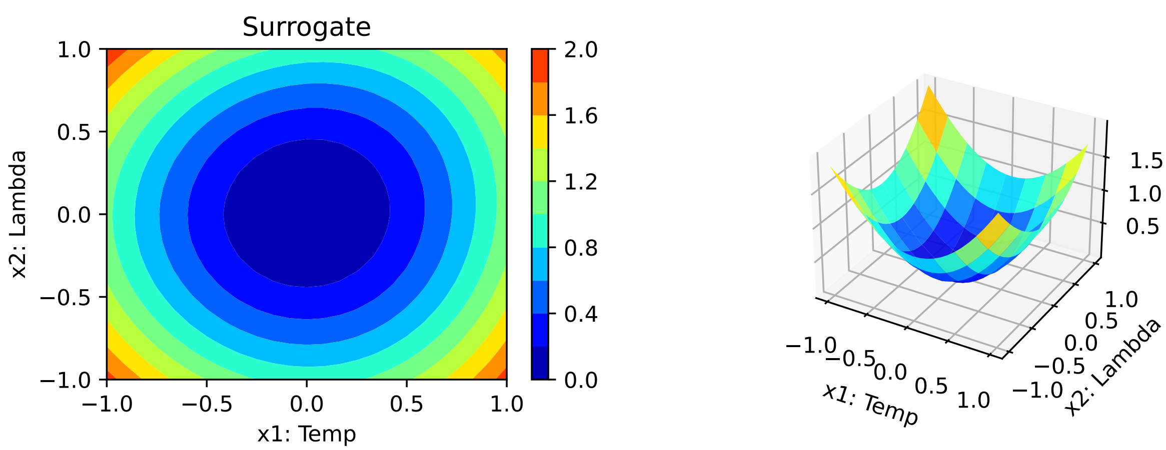
- In a similar manner, we can plot dimension $i = 0$ and $j = 2$:

```
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 $0 \le x \le 1$ 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 $-5 \le x \le 5$ 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 $-5 \le x \le 5$ 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 0x103ff1780>
```

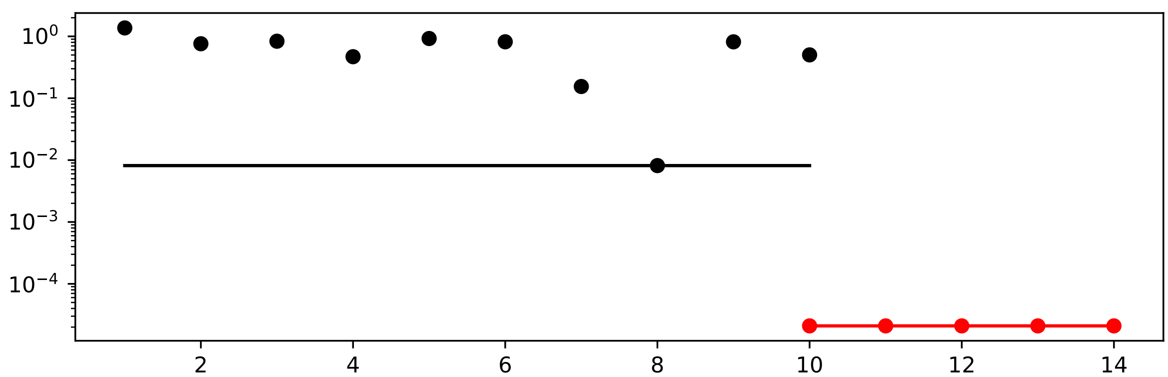
3.1.2 Results

```
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 0x167785fc0>
```

3.2.1 Taking a Look at the `theta` Values

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

```
spot_2_anisotropic.surrogate.theta
```

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

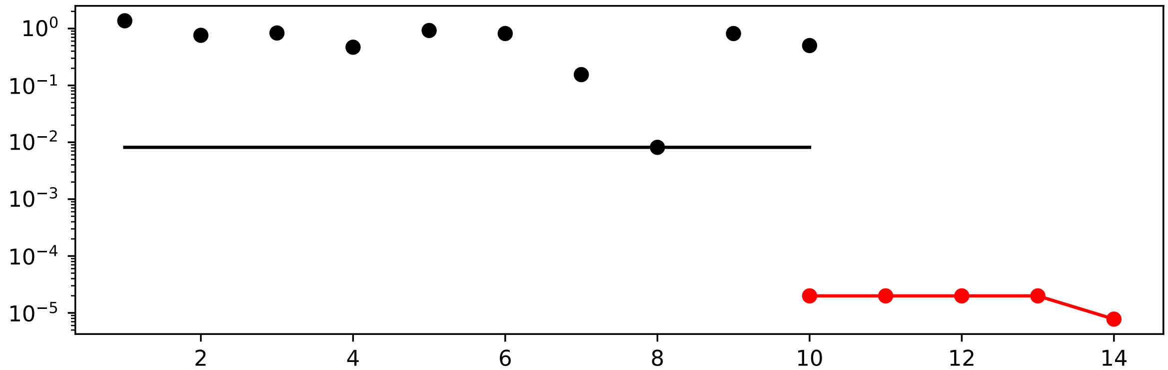
- Since the surrogate from the isotropic setting was stored as `spot_2`, we can also take a look at the `theta` value from this model:

```
spot_2.surrogate.theta
```

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

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

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

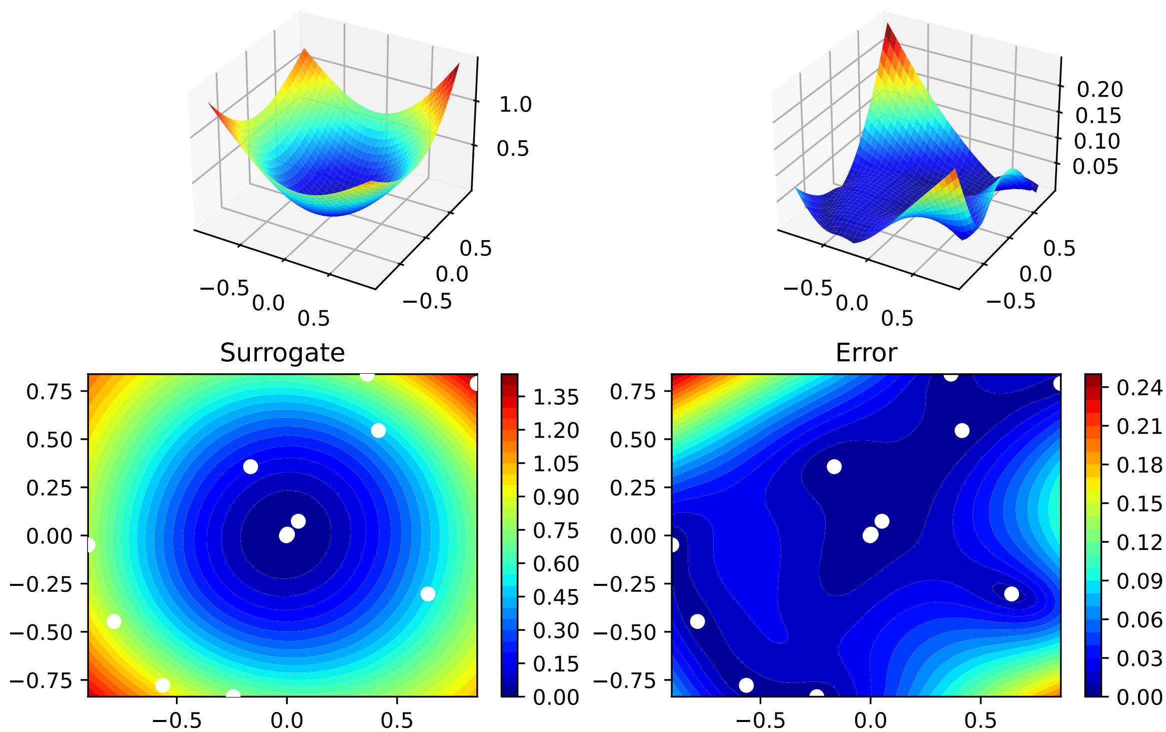


```
spot_2_anisotropic.print_results()
```

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

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

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



3.3 Exercises

3.3.1 `fun_branin`

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

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

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

3.3.2 `fun_sin_cos`

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

3.3.3 `fun_runge`

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

3.3.4 `fun_wingwt`

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

4 Using `sklearn` Surrogates in `spotPython`

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

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

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

4.1.1 The Objective Function Branin

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

```
y = a * (x2 - b * x1**2 + c * x1 - r) ** 2 + s * (1 - t) * np.cos(x1) + s,
where values of a, b, c, r, s and t are: a = 1, b = 5.1 / (4*pi**2),
c = 5 / pi, r = 6, s = 10 and t = 1 / (8*pi).
```

- It has three global minima:

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

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

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

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

4.1.2 Running the surrogate model based optimizer `Spot`:

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

```
spot_2.run()
```

```
<spotPython.spot.spot.Spot at 0x169864bb0>
```

4.1.3 Print the Results

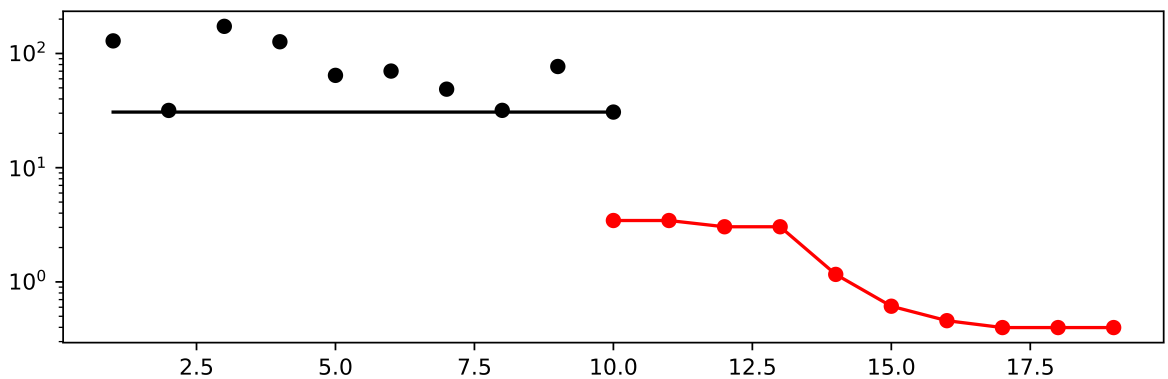
```
spot_2.print_results()
```

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

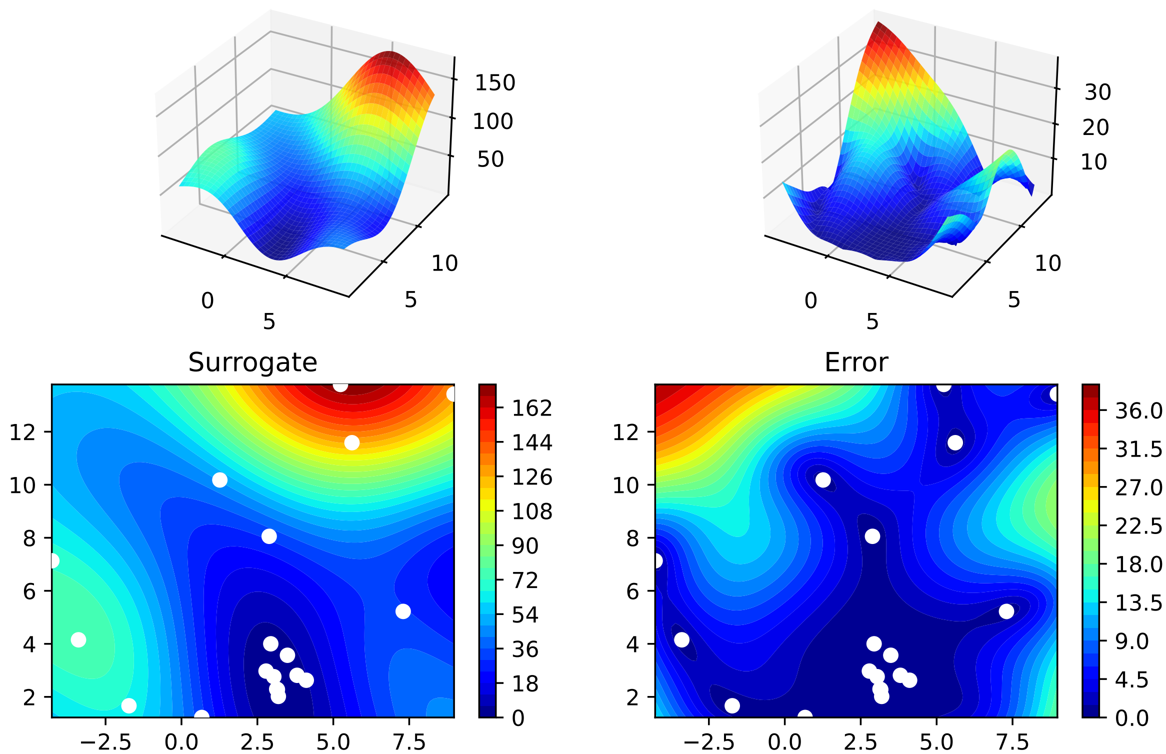
```
[['x0', 3.135528626303215], ['x1', 2.2926027772585886]]
```

4.1.4 Show the Progress and the Surrogate

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




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



4.2 Example: Using Surrogates From scikit-learn

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

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

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

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

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

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

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

4.2.1 GaussianProcessRegressor as a Surrogate

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

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

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

  `surrogate = S_GP`

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

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

```
True
```

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

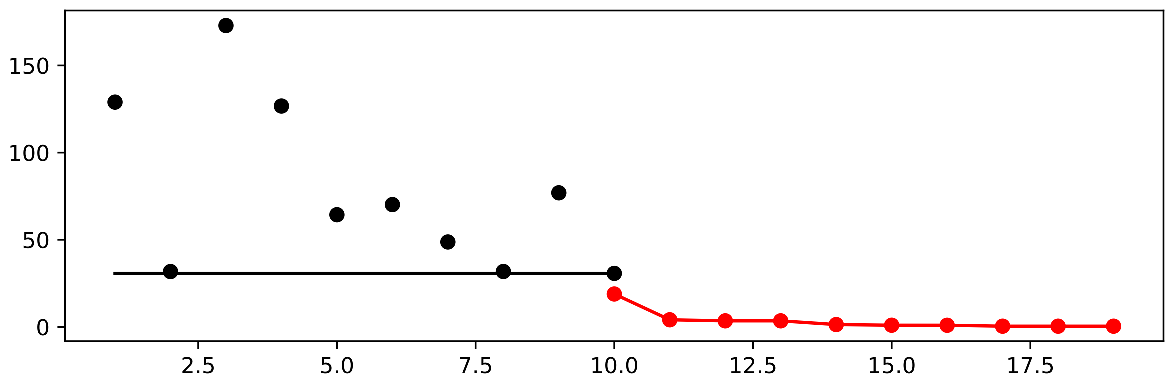
```
True
```

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

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

```
<spotPython.spot.spot.Spot at 0x16a329750>
```

```
spot_2_GP.plot_progress()
```




```
spot_2_GP.print_results()
```

```
min y: 0.398252885916484444
x0: 3.150311517377358
x1: 2.2689593248609237
```

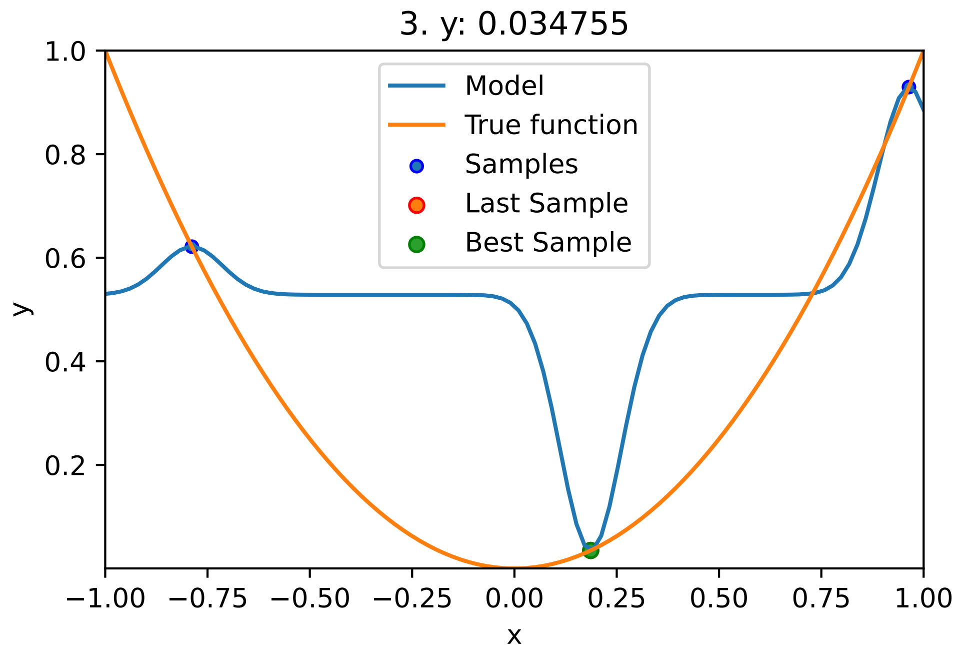
```
[['x0', 3.150311517377358], ['x1', 2.2689593248609237]]
```

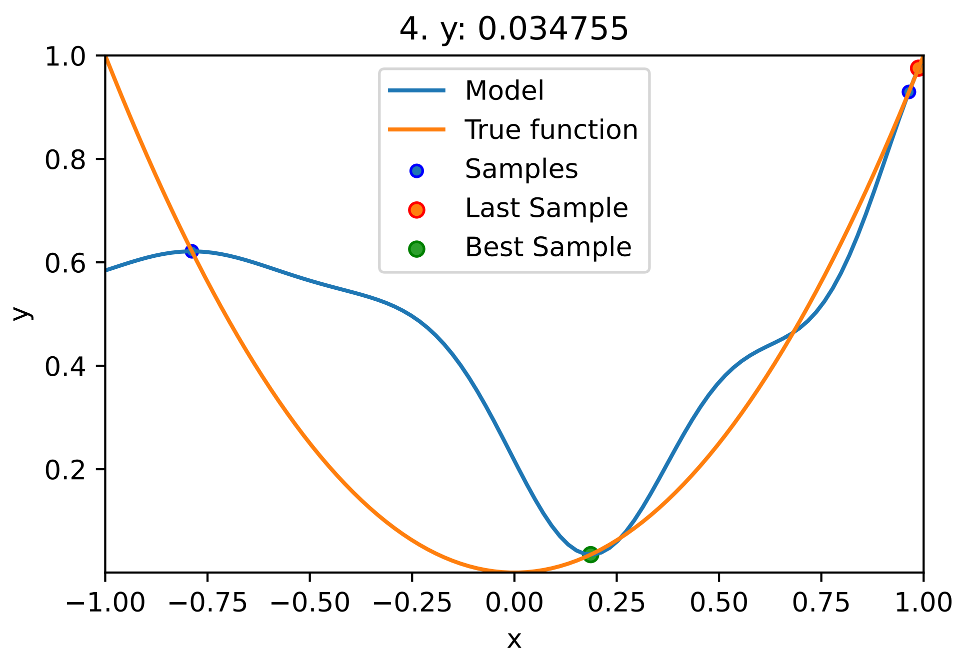
4.3 Example: One-dimensional Sphere Function With `spotPython`'s Kriging

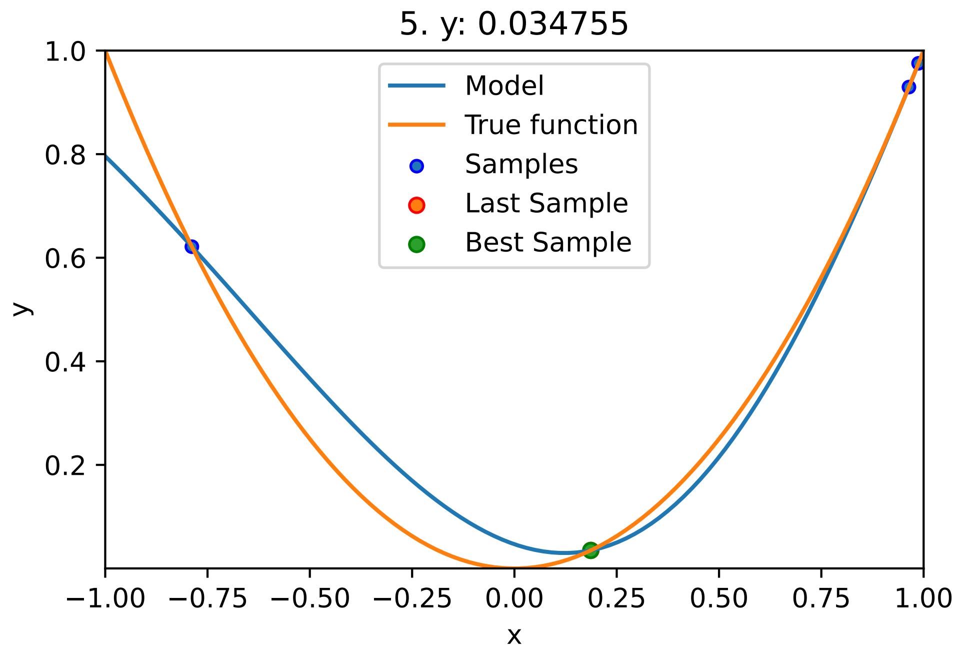
- In this example, we will use an one-dimensional function, which allows us to visualize the optimization process.
  - `show_models= True` is added to the argument list.

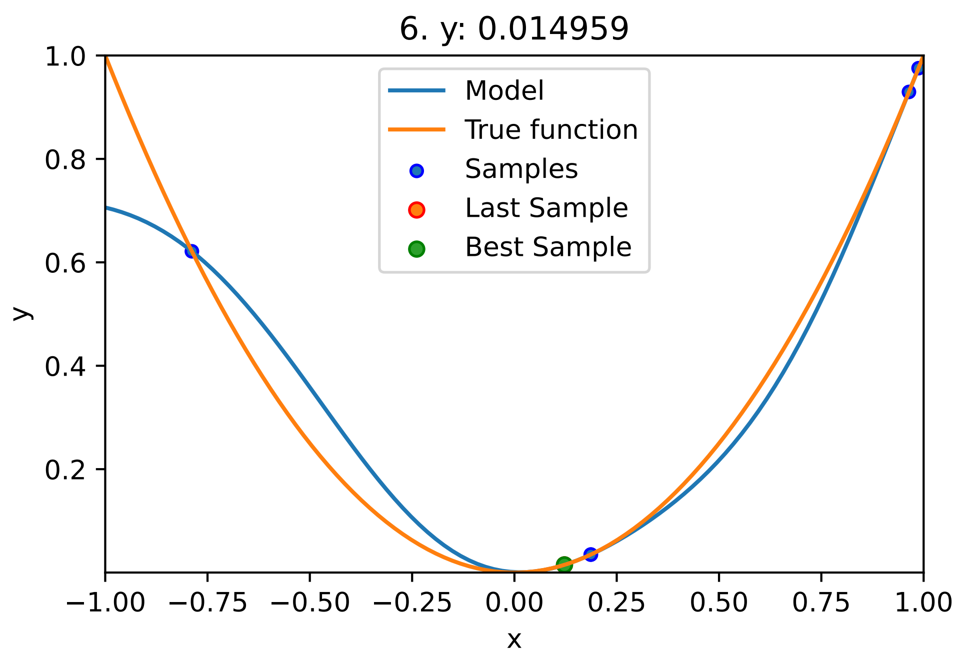
```
from spotPython.fun.objectivefunctions import analytical
lower = np.array([-1])
upper = np.array([1])
fun = analytical(seed=123).fun_sphere
```

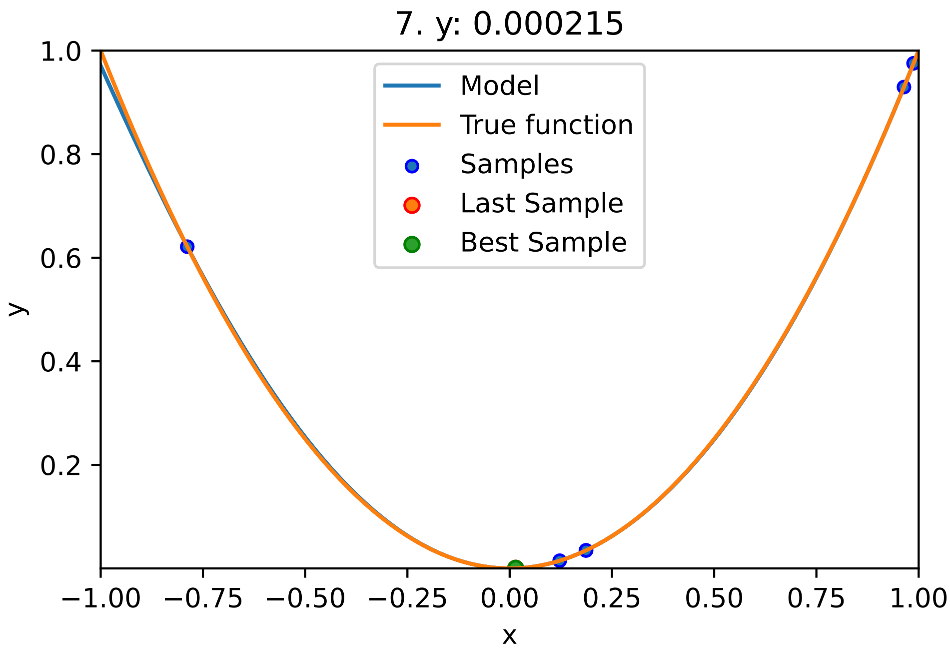
```
spot_1 = spot.Spot(fun=fun,
                   lower = lower,
                   upper = upper,
                   fun_evals = 10,
                   max_time = inf,
                   seed=123,
                   show_models= True,
                   tolerance_x = np.sqrt(np.spacing(1)),
      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


3. y: 0.034755
Model
True function
Samples
Last Sample
Best Sample
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

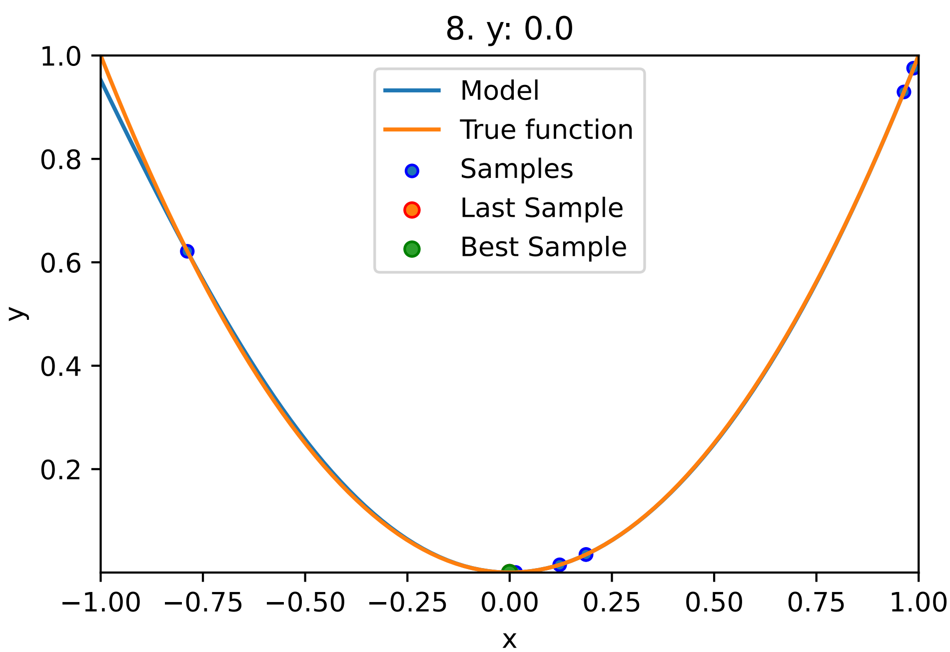

4. y: 0.034755
Model
True function
Samples
Last Sample
Best Sample
1.0
0.8
0.6
0.4
0.2
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x


5. y: 0.034755
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

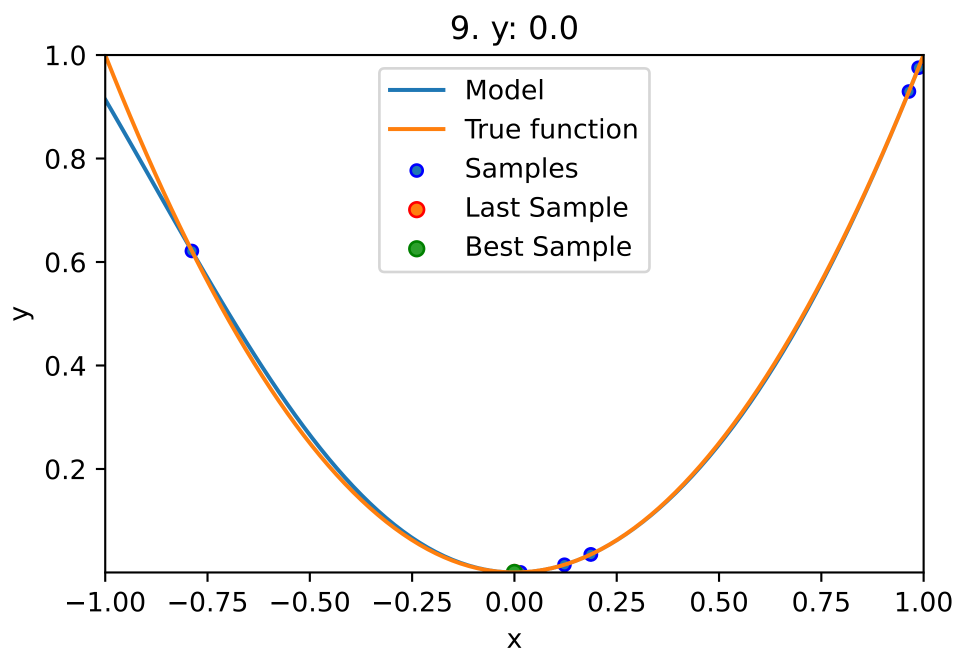

6. y: 0.014959
Model
True function
Samples
Last Sample
Best Sample
1.0
0.8
0.6
0.4
0.2
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x

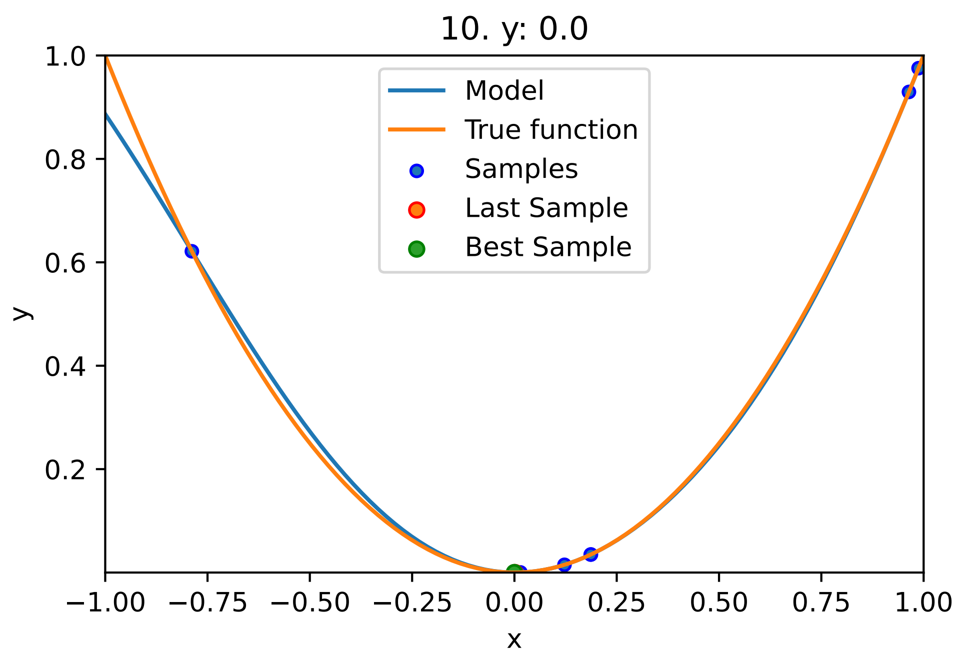


7. y: 0.000215
Model
True function
Samples
Last Sample
Best Sample
1.0
0.8
0.6
0.4
0.2
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x



8. 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






```
<spotPython.spot.spot.Spot at 0x177b4c1f0>
```

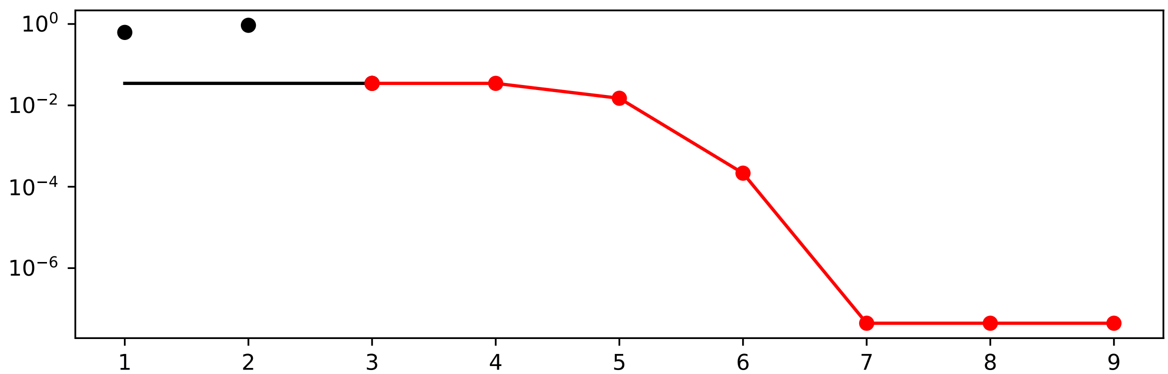
4.3.1 Results

```
spot_1.print_results()
```

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

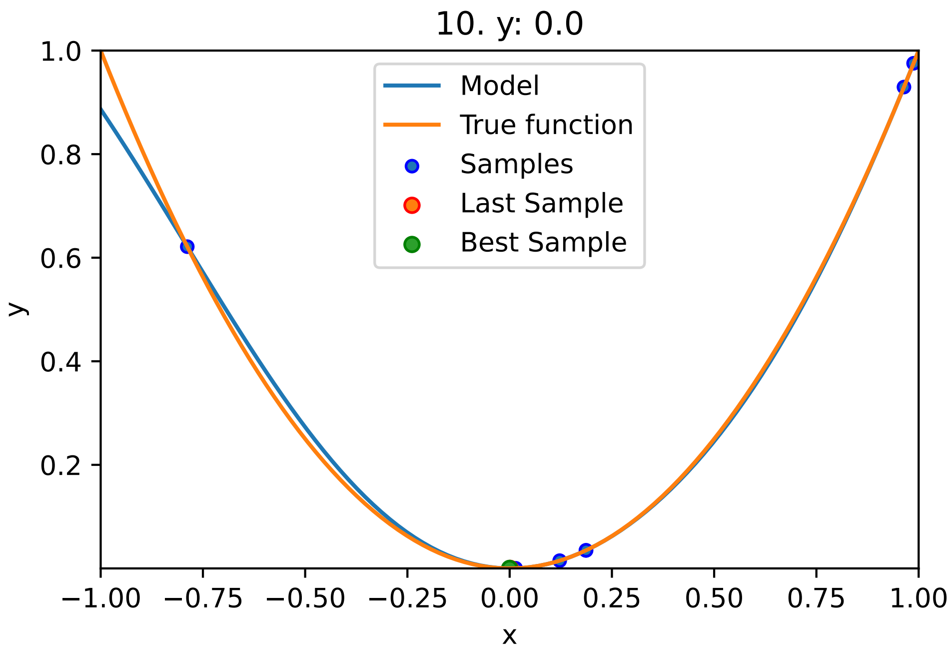
```
[['x0', -0.00021022017702259125]]
```

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



- The method `plot_model` plots the final surrogate:

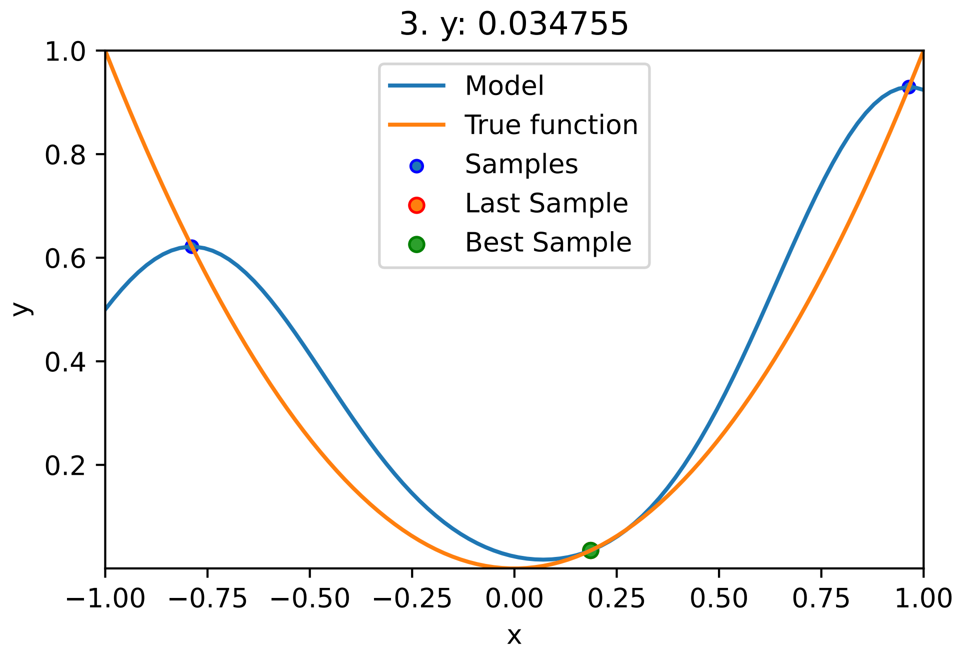
```
spot_1.plot_model()
```

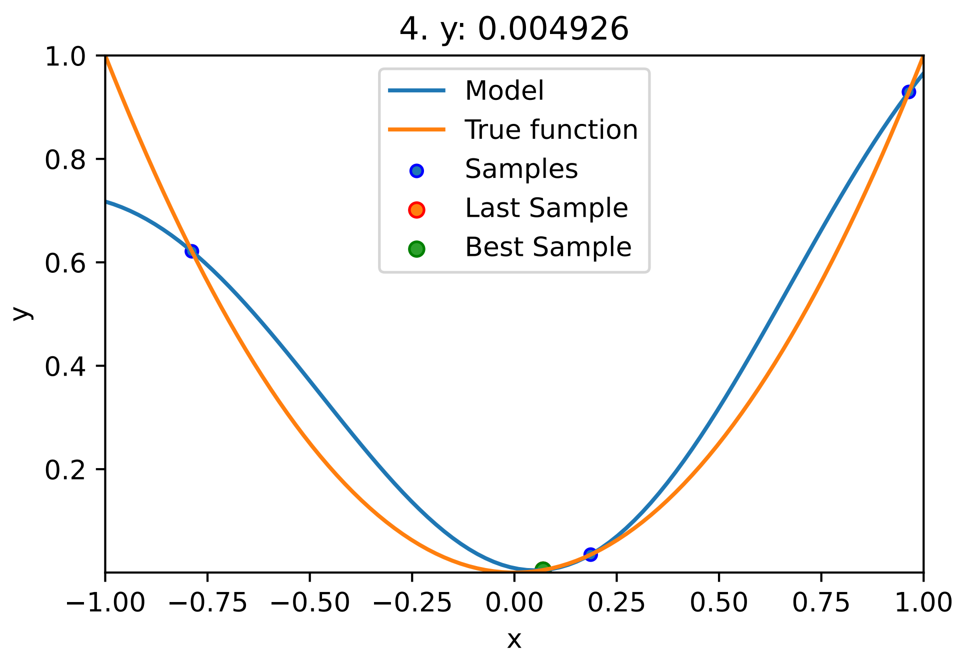


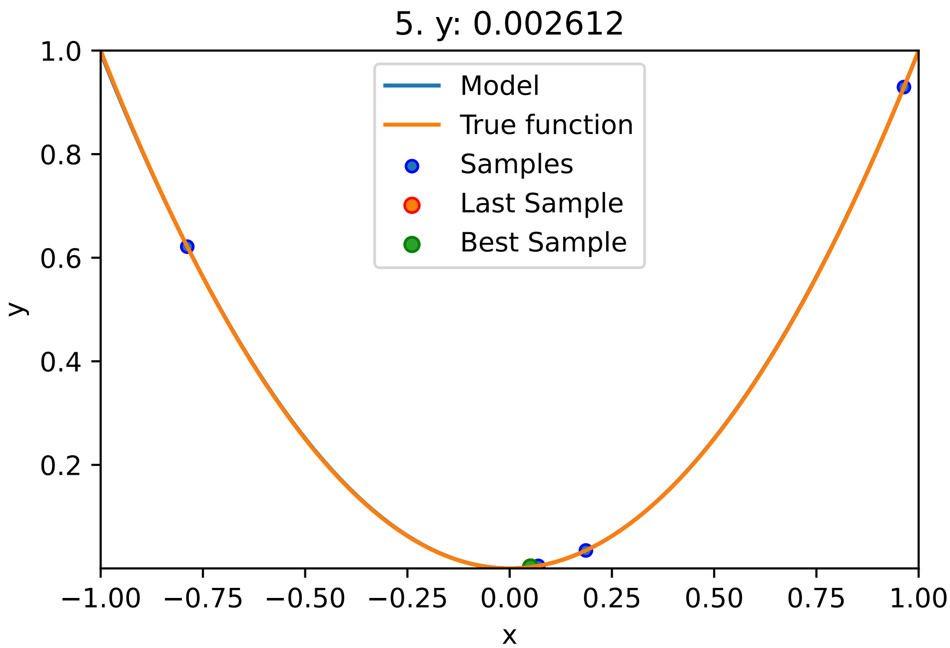
4.4 Example: `Sklearn` Model GaussianProcess

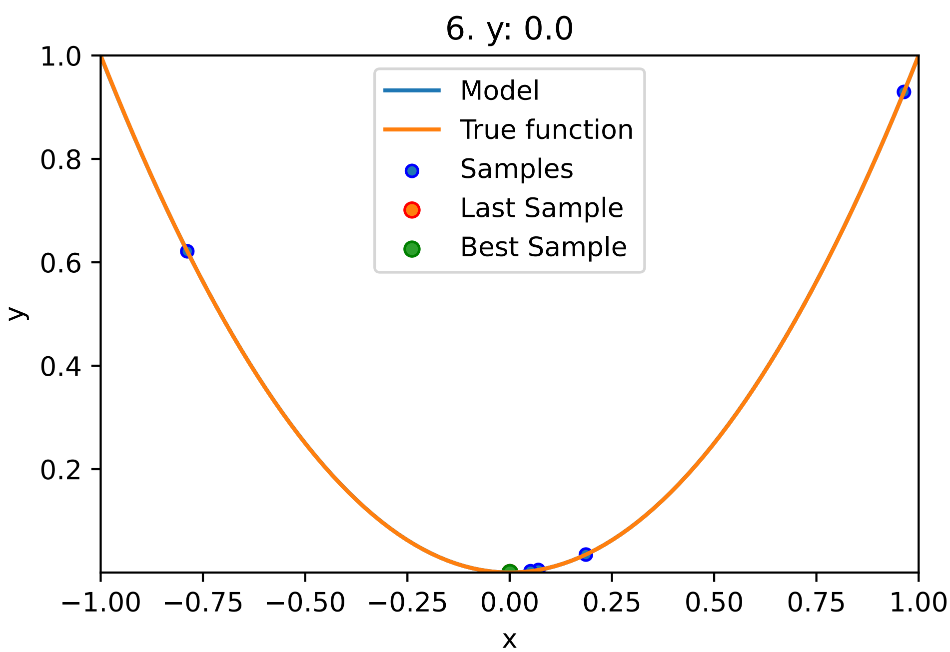
- This example visualizes the search process on the `GaussianProcessRegression` surrogate from `sklearn`.
- Therefore `surrogate = S_GP` is added to the argument list.

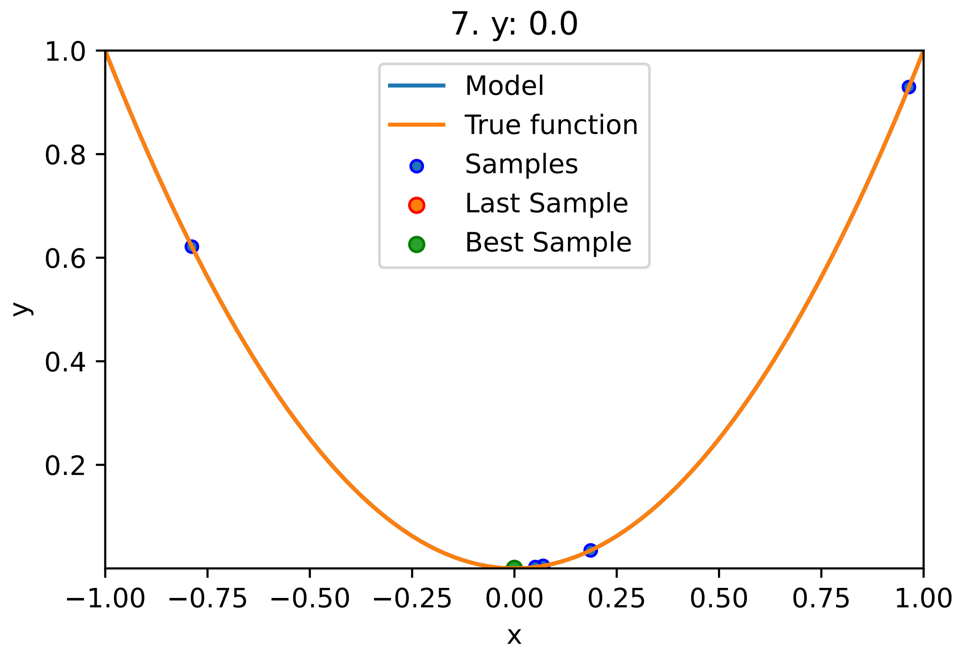
```
fun = analytical(seed=123).fun_sphere
spot_1_GP = spot.Spot(fun=fun,
                   lower = lower,
                   upper = upper,
                   fun_evals = 10,
                   max_time = inf,
                   seed=123,
                   show_models= True,
                   design_control={"init_size":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

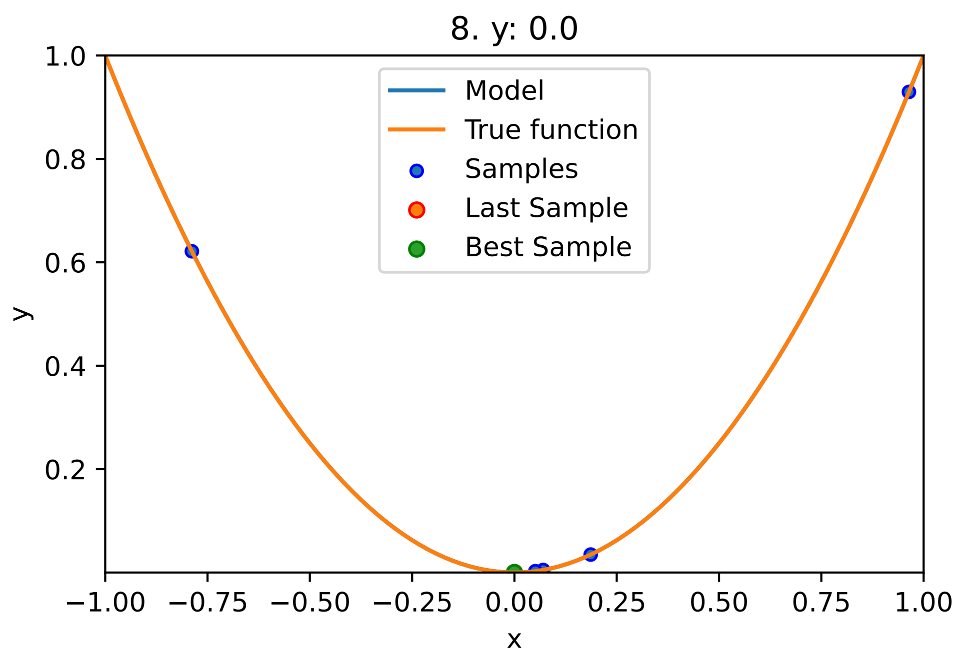

3. y: 0.034755
Model
True function
Samples
Last Sample
Best Sample
1.0
0.8
0.6
0.4
0.2
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x

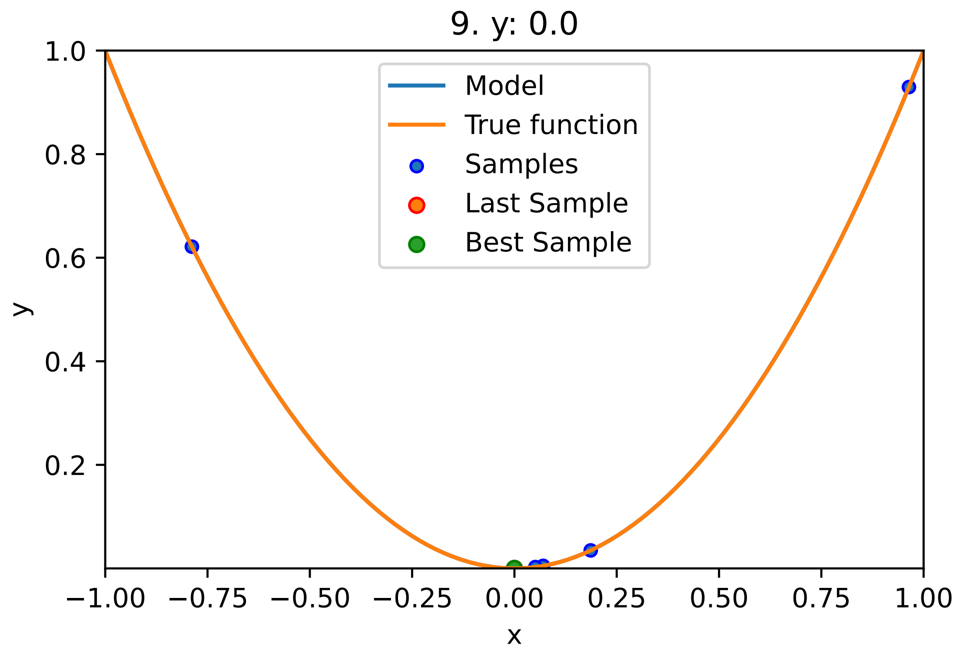

4. y: 0.004926
Model
True function
Samples
Last Sample
Best Sample
1.0
0.8
0.6
0.4
0.2
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x

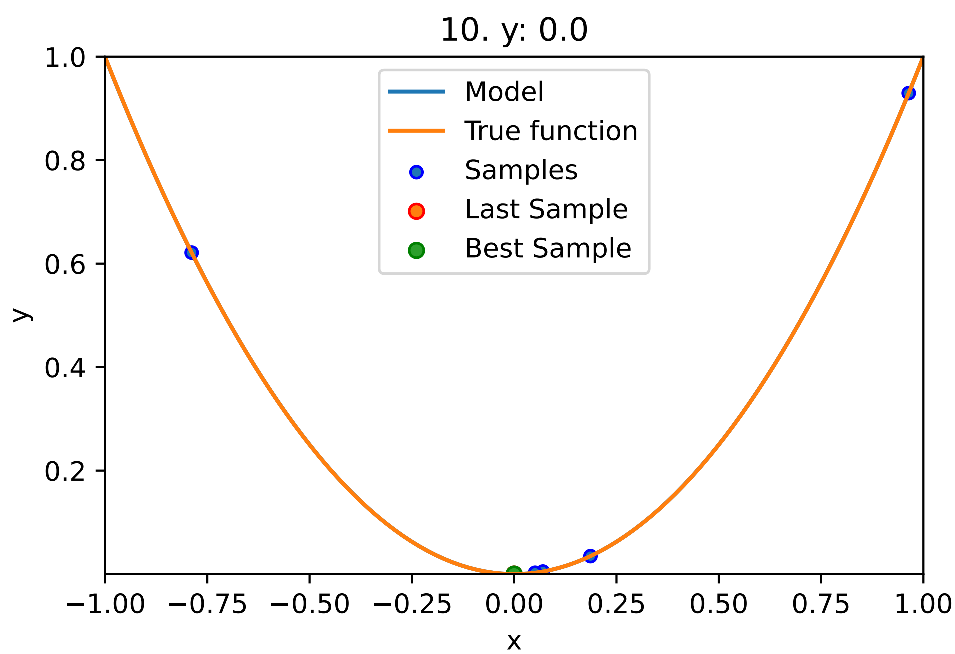

5. y: 0.002612
Model
True function
Samples
Last Sample
Best Sample
1.0
0.8
0.6
0.4
0.2
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x


6. 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. 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


8. 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





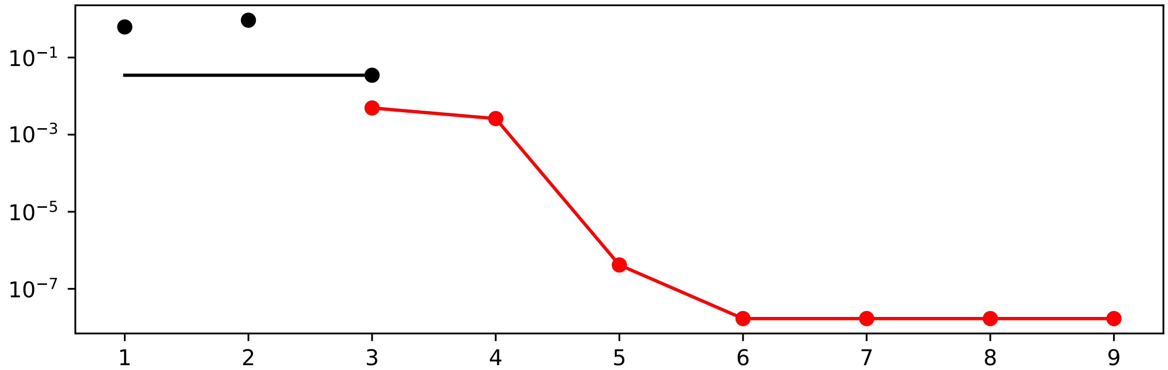

```
<spotPython.spot.spot.Spot at 0x2a4700670>
```

```
spot_1_GP.print_results()
```

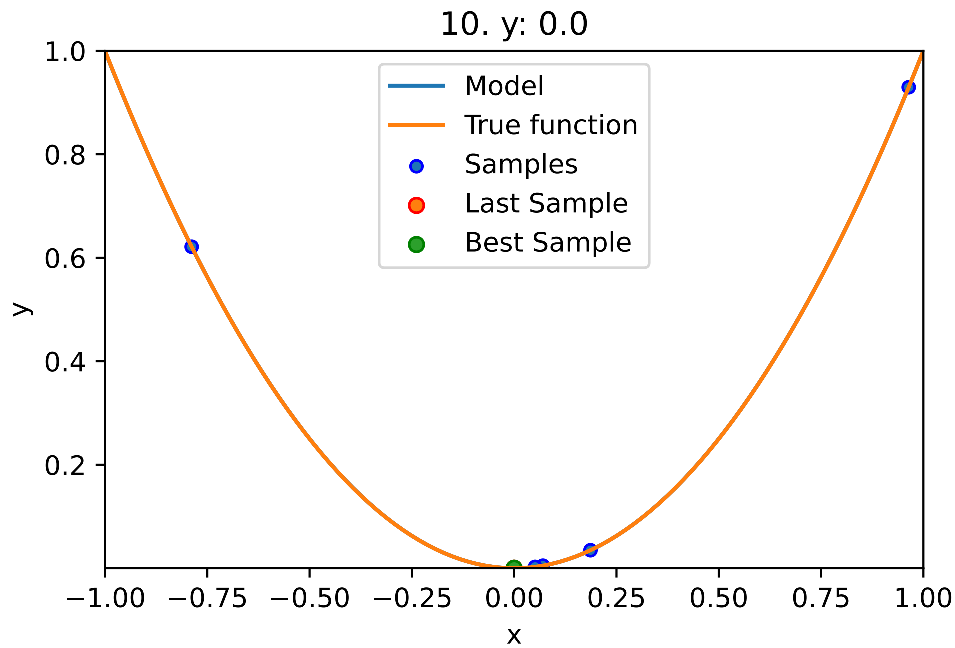
```
min y: 1.695790032115248e-08
x0: -0.00013022250312888506
```

```
[['x0', -0.00013022250312888506]]
```

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




```
spot_1_GP.plot_model()
```



4.5 Exercises

4.5.1 `DecisionTreeRegressor`

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

4.5.2 `RandomForestRegressor`

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

4.5.3 `linear_model.LinearRegression`

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

4.5.4 `linear_model.Ridge`

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

4.6 Exercise 2

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

5 Sequential Parameter Optimization: Using `scipy` Optimizers

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

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

5.1 The Objective Function Branin

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

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

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

- It has three global minima:

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

- Input Domain: This function is usually evaluated on the square x1 in [-5, 10] x x2 in [0, 15].

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

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

5.2 The Optimizer

- Differential Evalution from the `scikit.optimize` package, see https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.differential_evolution.html#scipy.optimize.differential_evolution is the default optimizer for the search on the surrogate.
- Other optimiers that are available in `spotPython`:
  - `dual_annealing`
  - `direct`
  - `shgo`
  - `basinhopping`, see https://docs.scipy.org/doc/scipy/reference/optimize.html#global-optimization.
- These can be selected as follows:

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

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

```
                    show_models= False,
                    design_control={"init_size": 10},
                    surrogate_control={"n_theta": 2,
                                       "model_optimizer": differential_evolution,
                                       "model_fun_evals": 1000,
                                       })
spot_de.run()
```

```
<spotPython.spot.spot.Spot at 0x1109818a0>
```

5.3 Print the Results

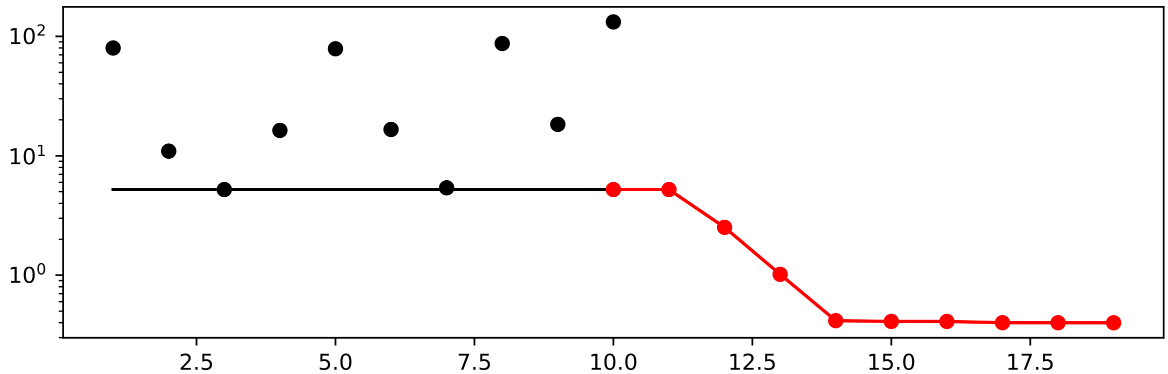
```
spot_de.print_results()
```

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

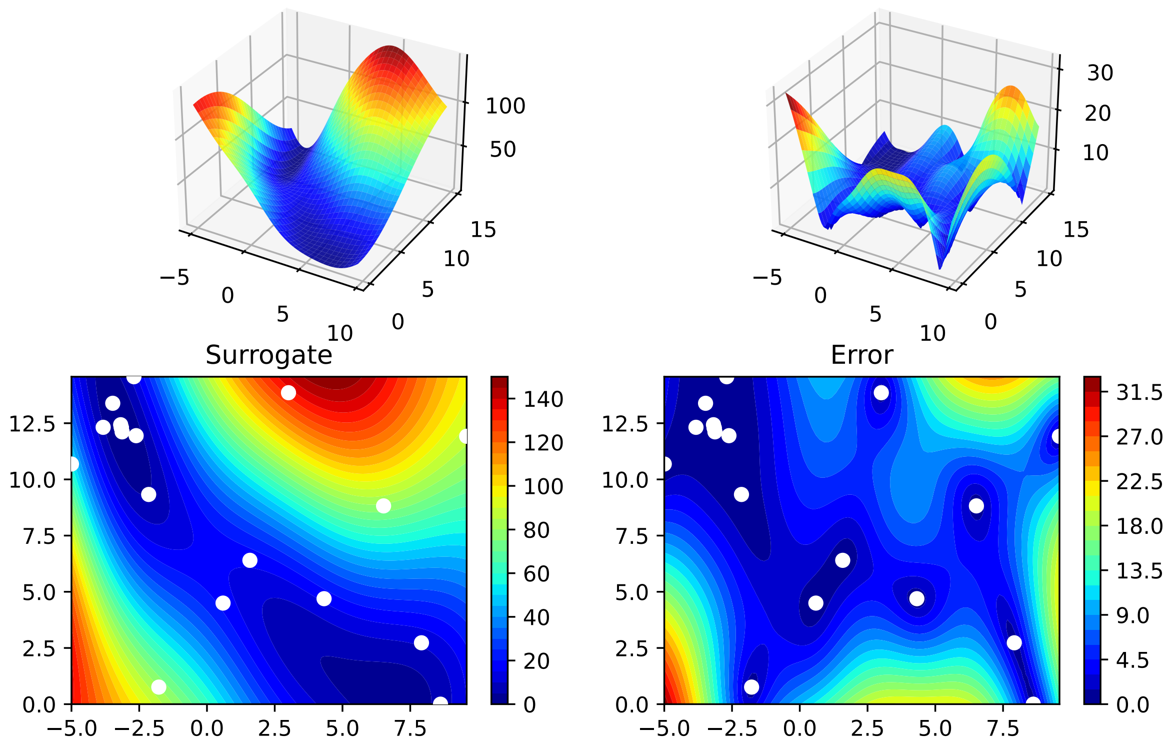
```
[['x0', -3.1570201165683587], ['x1', 12.289980569430284]]
```

5.4 Show the Progress

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



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



5.5 Exercises

5.5.1 dual_annealing

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

5.5.2 direct

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

5.5.3 `shgo`

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

5.5.4 `basinhopping`

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

5.5.5 Performance Comparison

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

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

The Branin function has three global minima:

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

- Which optima are found by the optimizers? Does the `seed` change this behavior?

6 Sequential Parameter Optimization: Gaussian Process Models

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

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

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

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

6.1.1 Train and Test Data

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

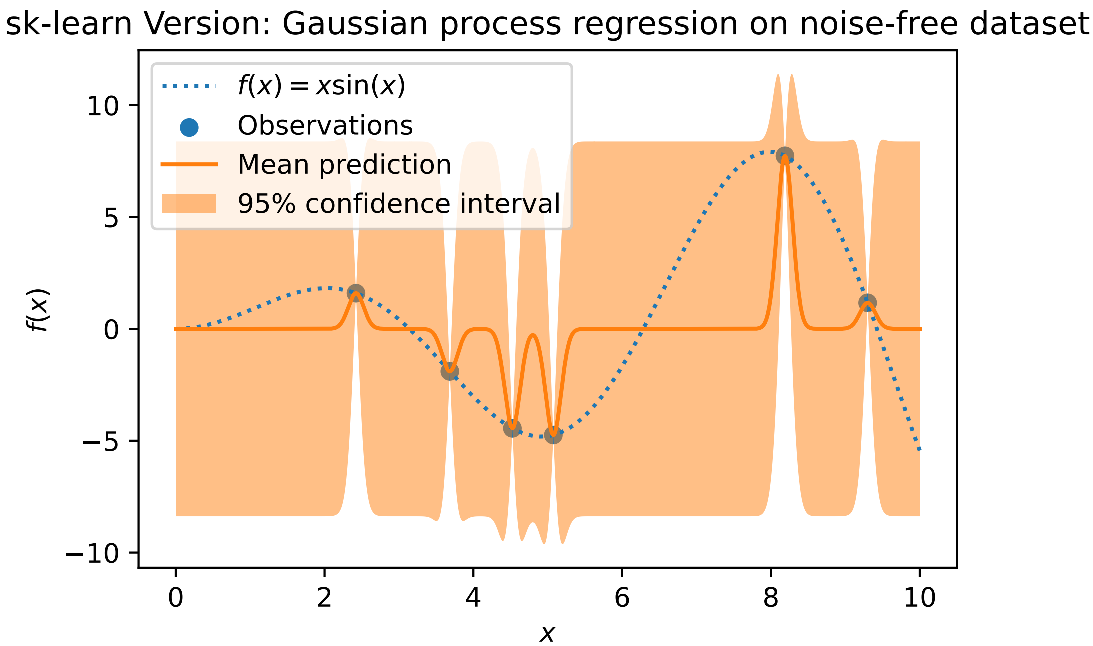
6.1.2 Building the Surrogate With `Sklearn`

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

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

6.1.3 Plotting the `Sklearn`Model

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



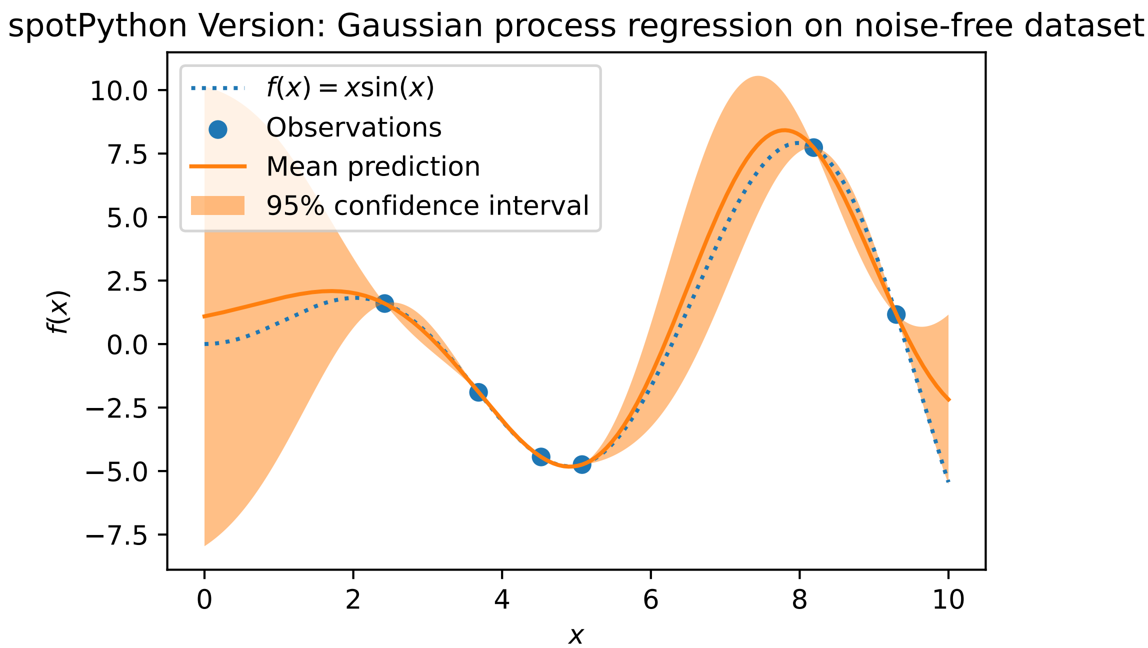
6.1.4 The `spotPython` Version

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

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

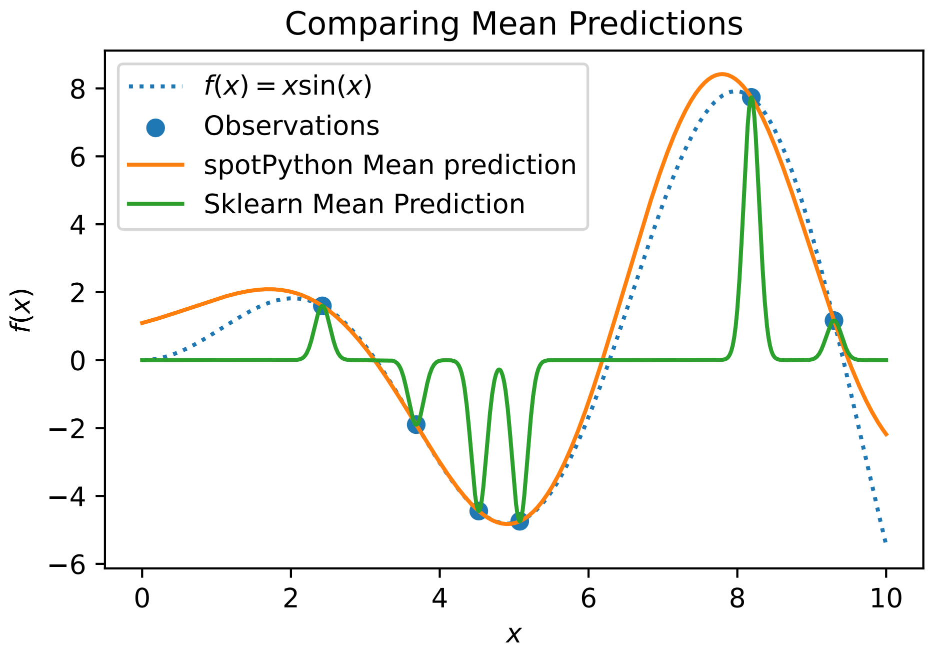
```
plt.plot(X, y, label=r"$f(x) = x \sin(x)$", linestyle="dotted")
plt.scatter(X_train, y_train, label="Observations")
plt.plot(X, S_mean_prediction, label="Mean prediction")
plt.fill_between(
    X.ravel(),
    S_mean_prediction - 1.96 * S_std_prediction,
    S_mean_prediction + 1.96 * S_std_prediction,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

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



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

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



6.2 Exercises

6.2.1 `Schonlau Example Function`

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

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

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

6.2.2 `Forrester Example Function`

- The Forrester Example Function is defined as follows:

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

- Data points are generated as follows:

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

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

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

6.2.3 `fun_runge` Function (1-dim)

- The Runge function is defined as follows:

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

- Data points are generated as follows:

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

- Describe the function.

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

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

6.2.4 `fun_cubed (1-dim)`

- The Cubed function is defined as follows:

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

- Data points are generated as follows:

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

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

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

6.2.5 The Effect of Noise

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

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

7 Expected Improvement

7.1 Example: `Spot` and the 1-dim Sphere Function

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

7.1.1 The Objective Function: 1-dim Sphere

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

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

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

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

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

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

```
spot_1.run()
```

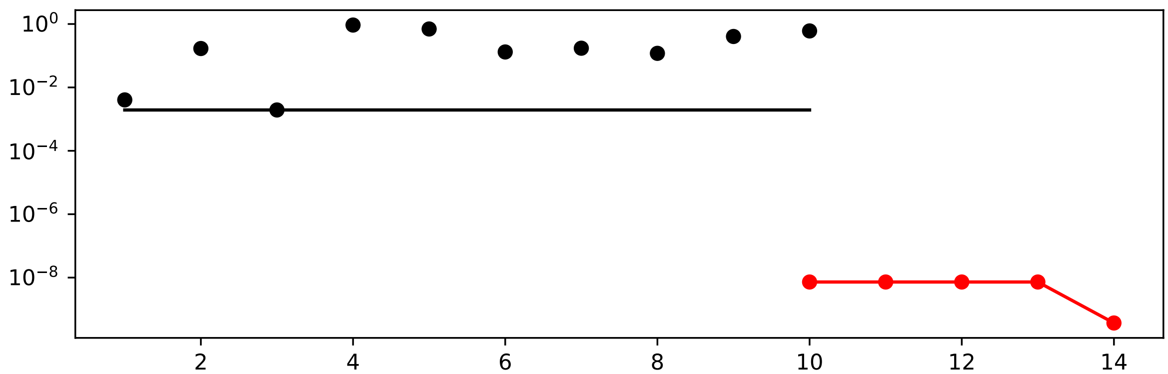
```
<spotPython.spot.spot.Spot at 0x136d1ceb0>
```

7.1.2 Results

```
spot_1.print_results()
```

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

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

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

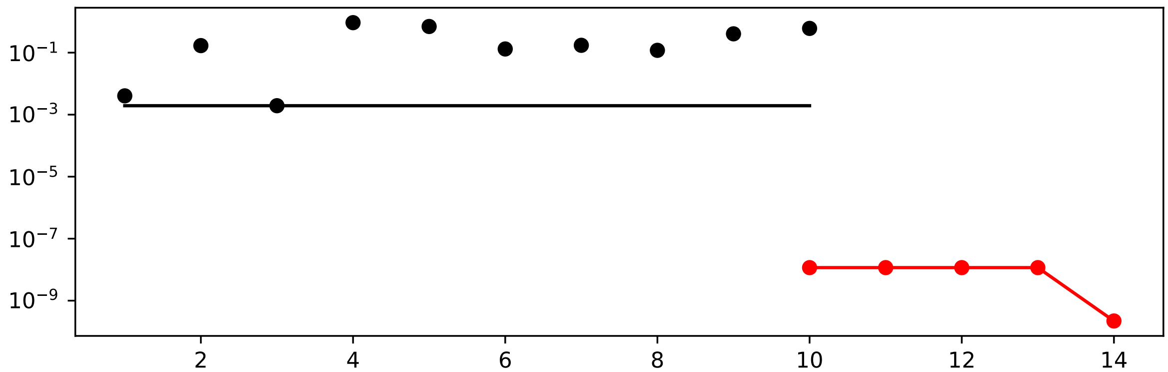


7.2 Same, but with EI as infill_criterion

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

```
<spotPython.spot.spot.Spot at 0x147168e20>
```

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




```
spot_1_ei.print_results()
```

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

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

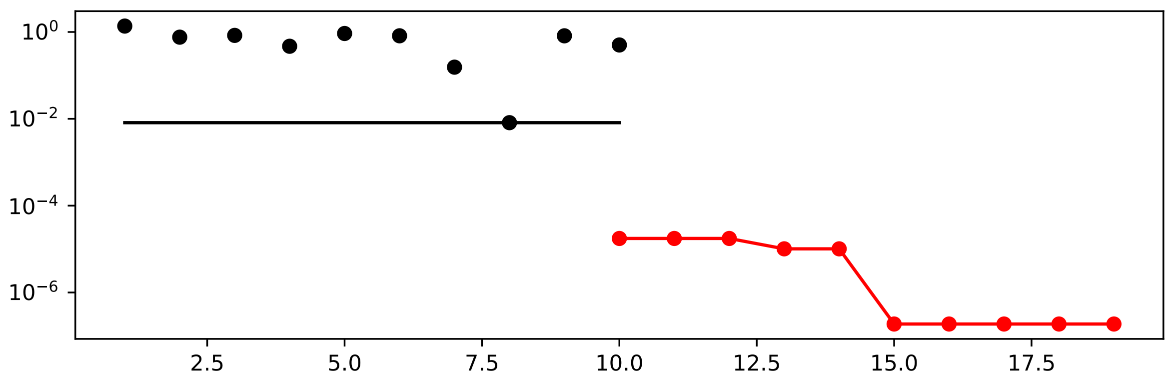
7.3 Non-isotropic Kriging

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

```
                     design_control={"init_size": 10,
                                     "repeats": 1},
                     surrogate_control={"noise": False,
                                        "cod_type": "norm",
                                        "min_theta": -4,
                                        "max_theta": 3,
                                        "n_theta": 2,
                                        "model_optimizer": differential_evolution,
                                        "model_fun_evals": 1000,
                                        })
spot_2_ei_noniso.run()
```

```
<spotPython.spot.spot.Spot at 0x1474c85b0>
```

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

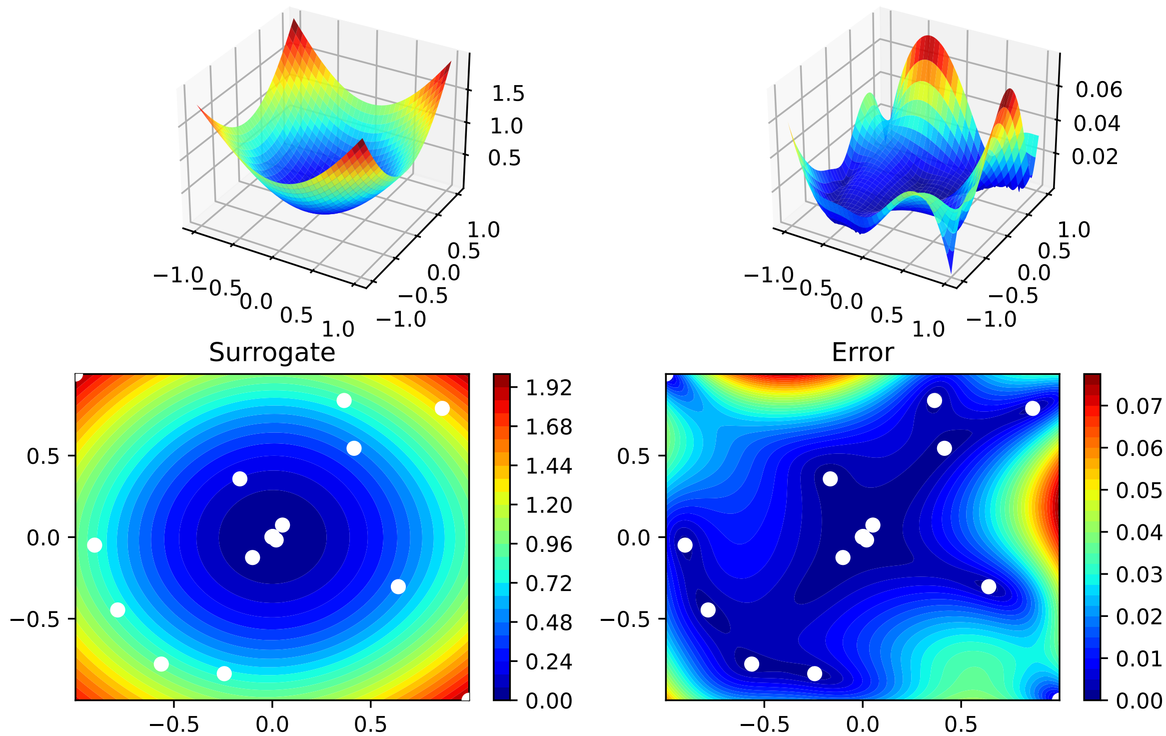



```
spot_2_ei_noniso.print_results()
```

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

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

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



7.4 Using `sklearn` Surrogates

7.4.1 The spot Loop

The `spot` loop consists of the following steps:

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

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

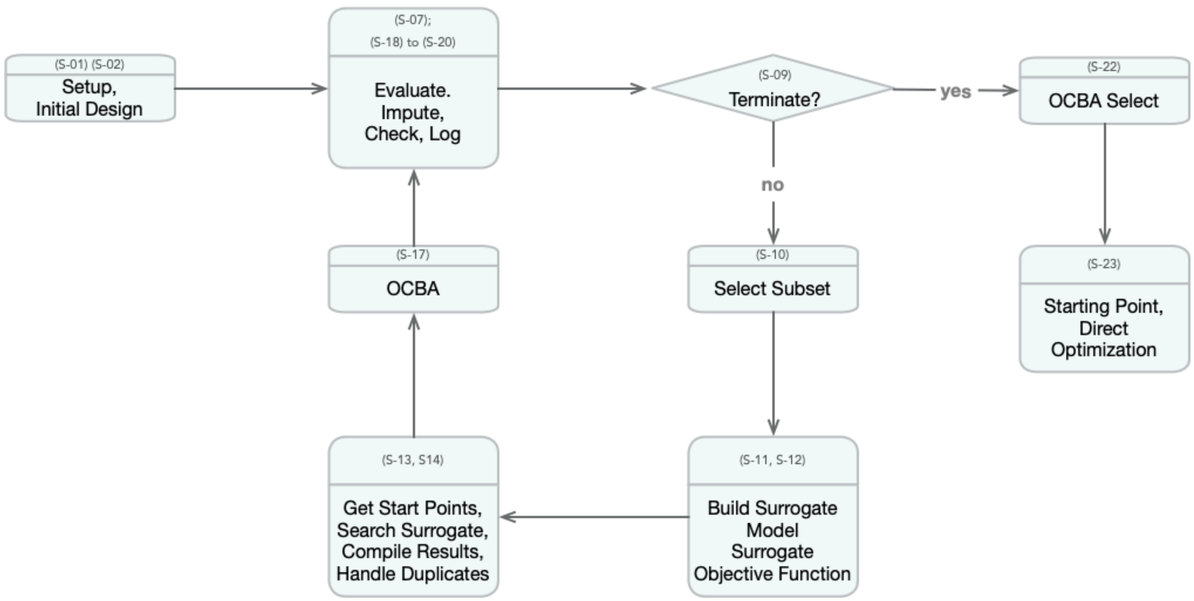


Figure 7.1: Visual representation of the model based search with SPOT. Taken from: Bartz-Beielstein, T., and Zaefferer, M. Hyperparameter tuning approaches. In Hyperparameter Tuning for Machine and Deep Learning with R - A Practical Guide, E. Bartz, T. Bartz-Beielstein, M. Zaefferer, and O. Mersmann,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

7.4.2 spot: The Initial Model

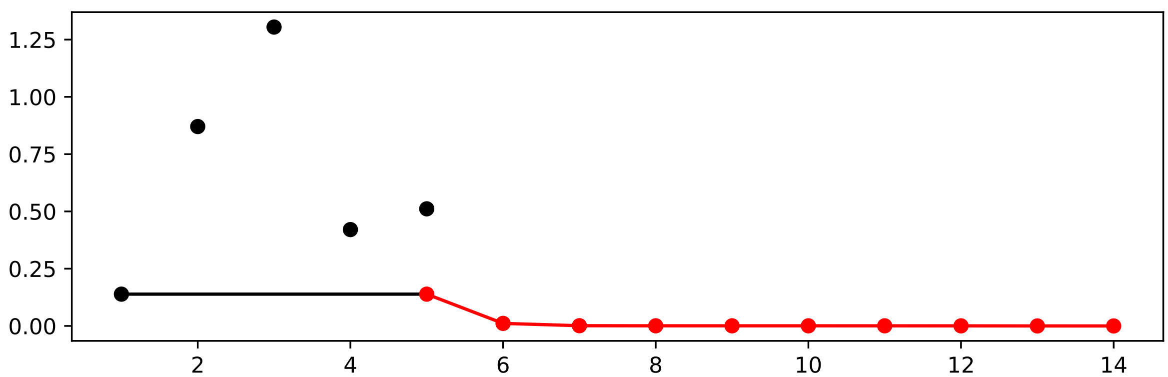
7.4.2.1 Example: Modifying the initial design size

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

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

```
<spotPython.spot.spot.Spot at 0x15f5c3820>
```

```
spot_ei.plot_progress()
```




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

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

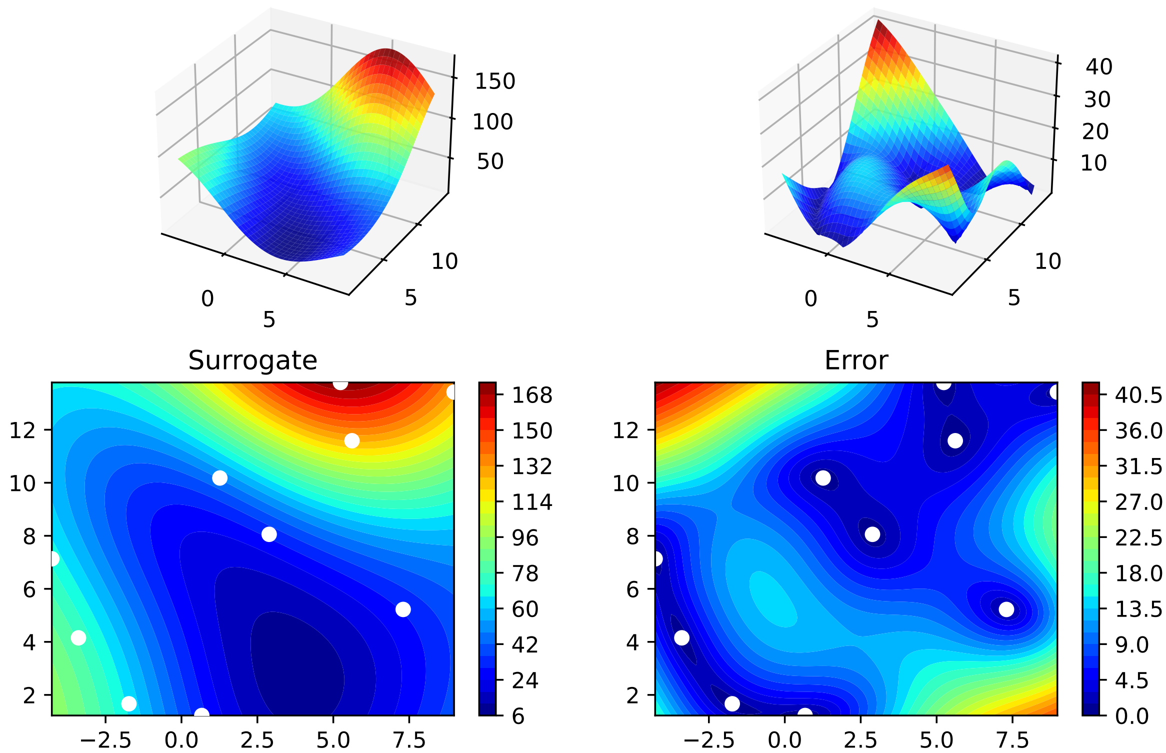
7.4.3 Init: Build Initial Design

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

```python
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]
```

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




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

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

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

7.4.4 Evaluate

7.4.5 Build Surrogate

7.4.6 A Simple Predictor

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

- Assume that only two (very costly) measurements are available:

  1. f(0) = 0.5
  2. f(2) = 2.5

- We are interested in the value at $x_0 = 1$, i.e., $f(x_0 = 1)$, but cannot run an additional, third experiment.

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

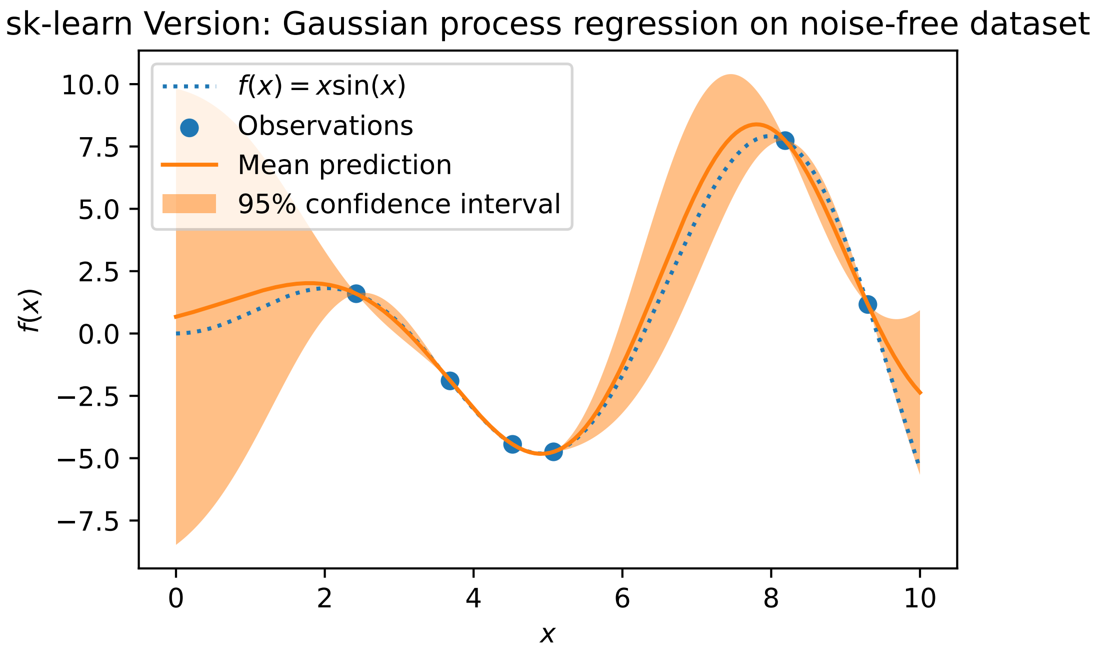
```
[1.5]
```

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

7.5 Gaussian Processes regression: basic introductory example

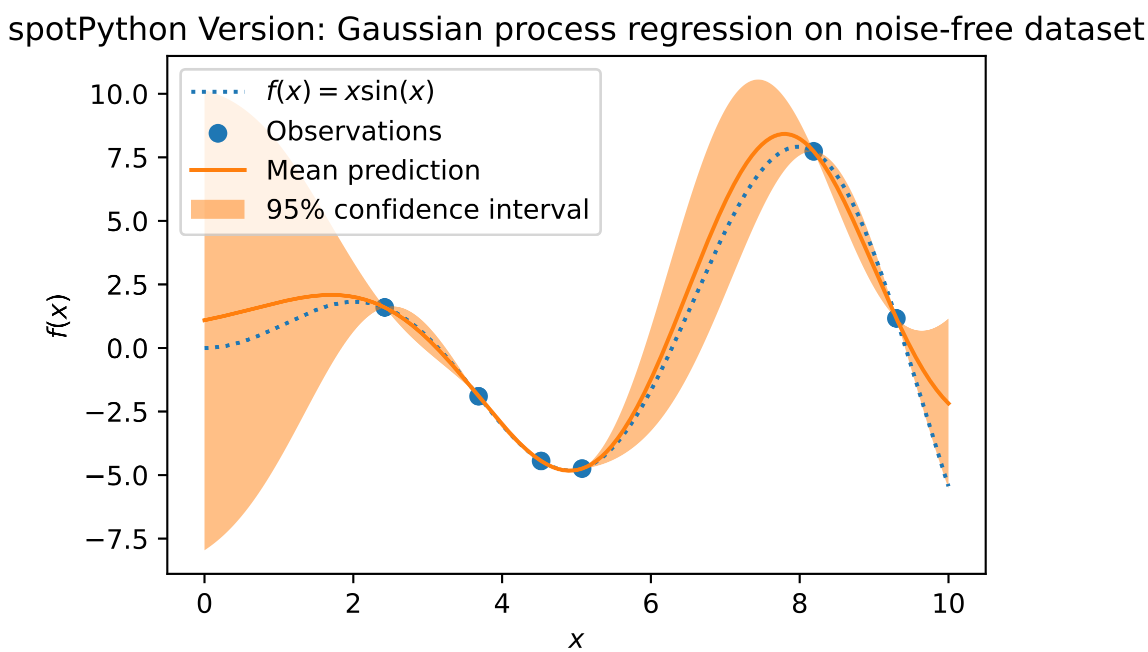
This example was taken from scikit-learn. After fitting our model, we see that the hyperparameters of the kernel have been optimized. Now, we will use our kernel to compute the mean prediction of the full dataset and plot the 95% confidence interval.

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

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

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

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

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




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


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

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

std_prediction

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

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



7.6 The Surrogate: Using scikit-learn models

Default is the internal `kriging` surrogate.

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

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

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

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

- and many more:

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

- The scikit-learn GP model `S_GP` is selected.

```python
S = S_GP
```

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

```
True
```

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

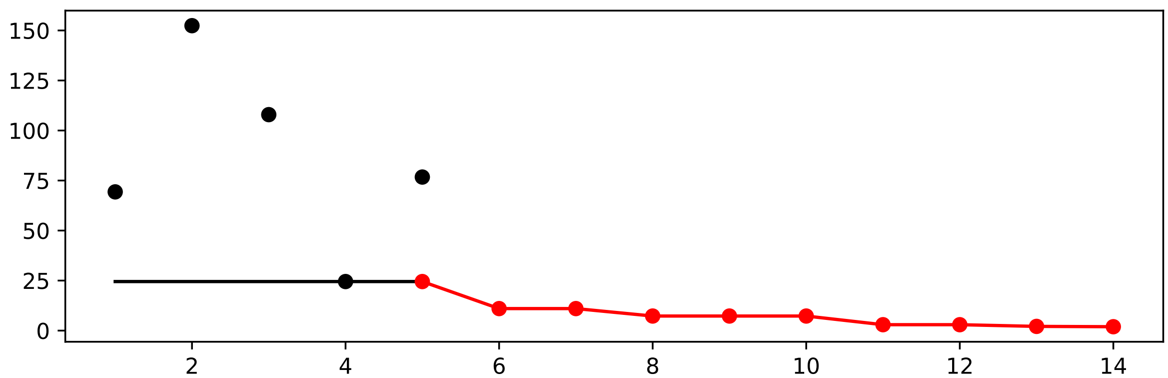
```
spot_GP.run()
```

```
<spotPython.spot.spot.Spot at 0x15f362dd0>
```

```
spot_GP.y
```

```
array([ 69.32459936, 152.38491454, 107.92560483,  24.51465459,
        76.73500031,  86.30426908,  11.00309249,  16.11743298,
         7.28107072,  21.82339972,  10.96088904,   2.9518833 ,
         3.02909864,   2.10497197,   1.94316172])
```

```
spot_GP.plot_progress()
```




```
spot_GP.print_results()
```

```
min y: 1.943161723768208
x0: 10.0
x1: 2.9983673915231734
```

```
[['x0', 10.0], ['x1', 2.9983673915231734]]
```

7.7 Additional Examples

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

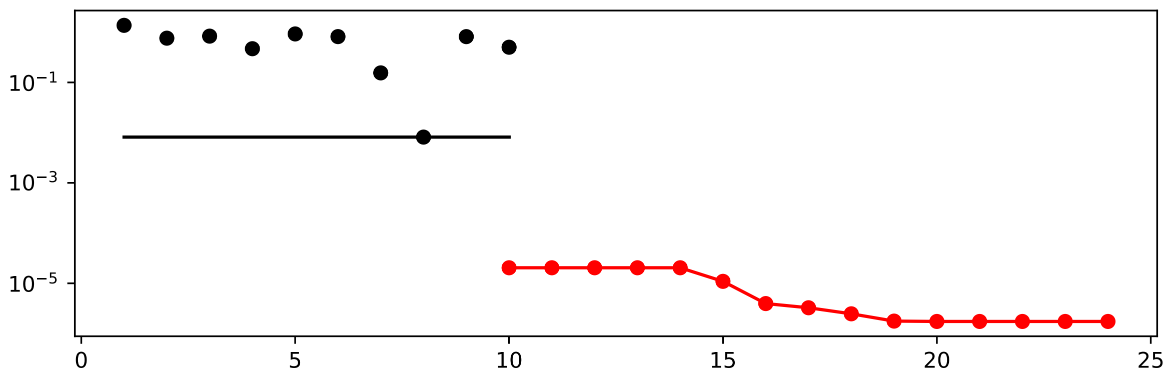
```
kernel = 1 * RBF(length_scale=1.0, length_scale_bounds=(1e-2, 1e2))
S_GP = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=9)
```

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

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

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

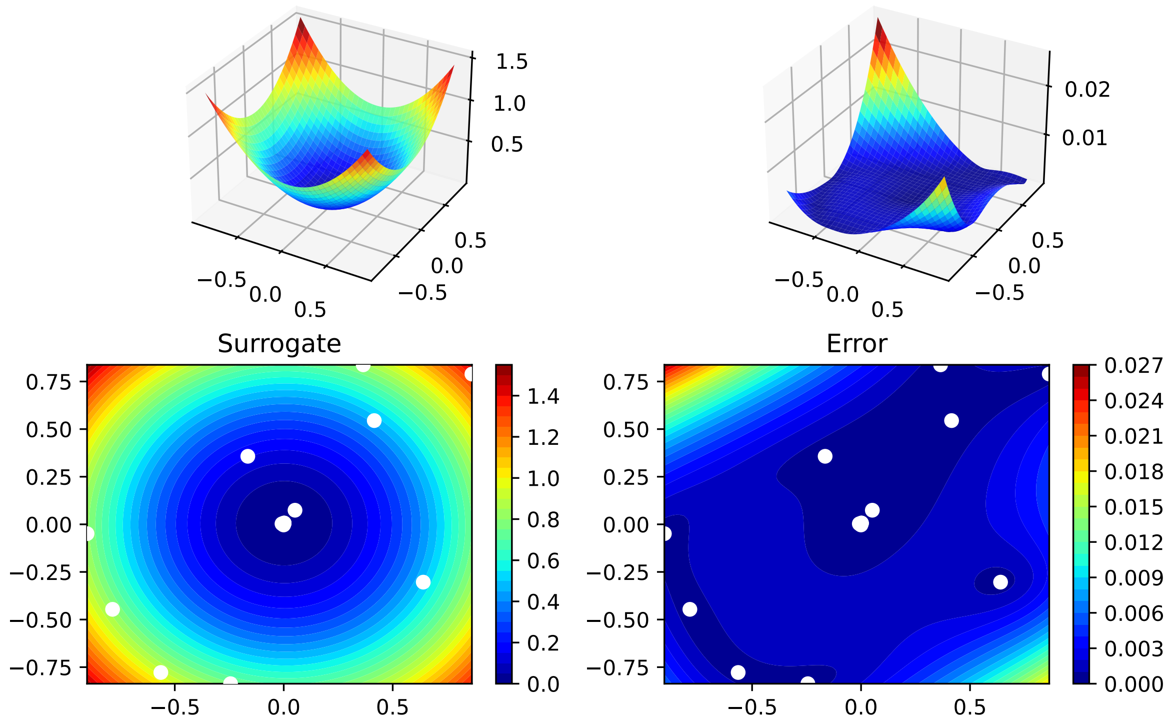
```
                                design_control=design_control,
                                surrogate_control=surrogate_control)

spot_S_K.run()
```

```
<spotPython.spot.spot.Spot at 0x16d661bd0>
```

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




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




```
spot_S_K.print_results()
```

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

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

7.7.1 Optimize on Surrogate

7.7.2 Evaluate on Real Objective

7.7.3 Impute / Infill new Points

7.8 Tests

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

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

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

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

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

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

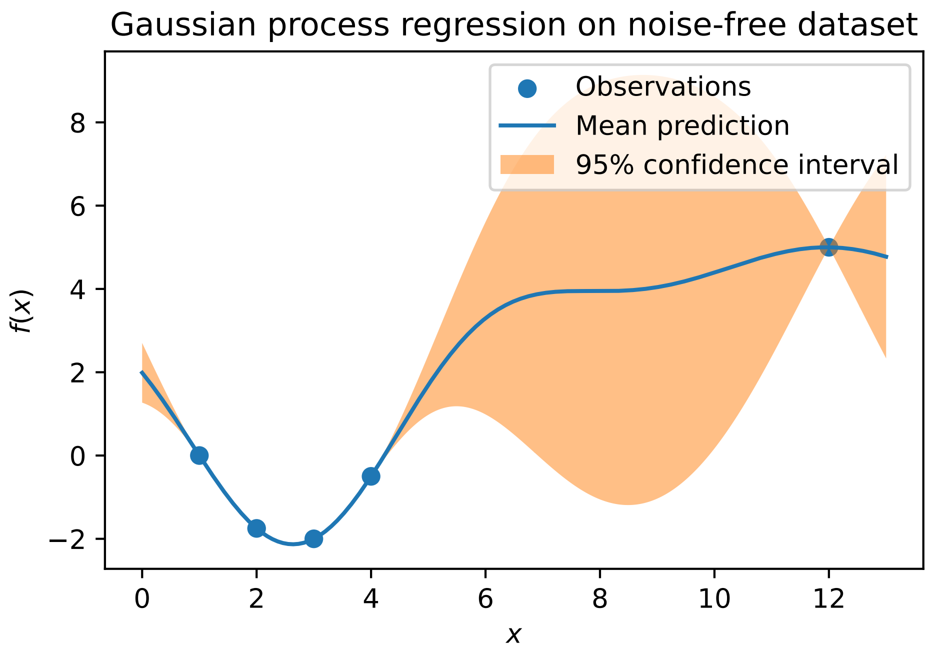
```
 [-0.16484832  0.35724741]
 [ 0.05170659  0.07401196]
 [-0.78548145 -0.44638164]
 [ 0.64017497 -0.30363301]]
[1.36857656 0.75992983 0.83463487 0.46918172 0.92329124 0.8170764
 0.15480068 0.00815134 0.81623768 0.502017  ]
[[0.00160553 0.00428429]
 [0.00160553 0.00428429]]
```

7.9 EI: The Famous Schonlau Example

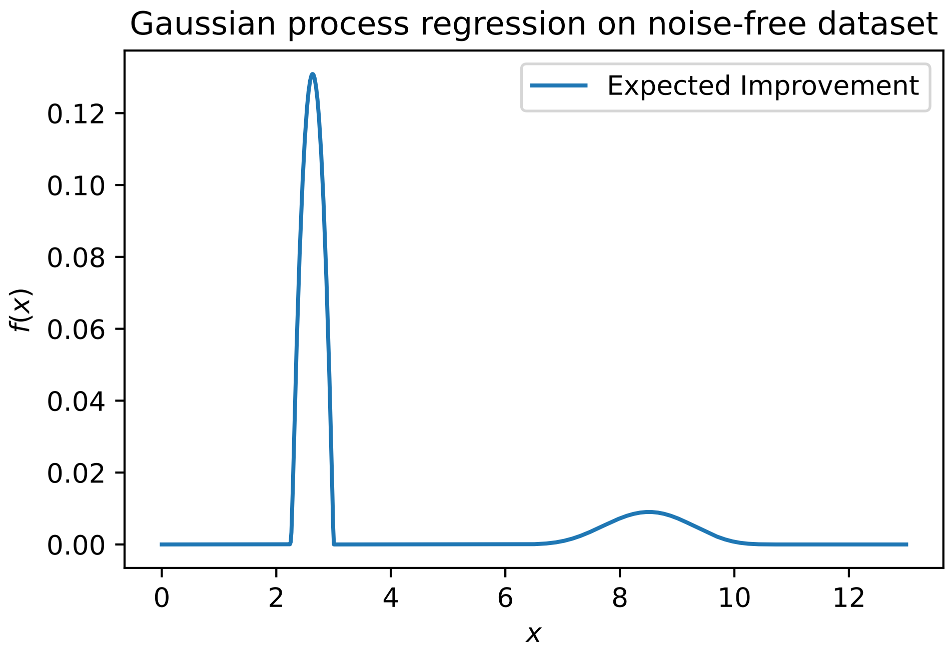
```
X_train0 = np.array([1, 2, 3, 4, 12]).reshape(-1,1)
X_train = np.linspace(start=0, stop=10, num=5).reshape(-1, 1)
```

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

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

S = Kriging(name='kriging',  seed=123, log_level=50, n_theta=1, noise=False, cod_type="nor
S.fit(X_train, y_train)

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

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

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




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



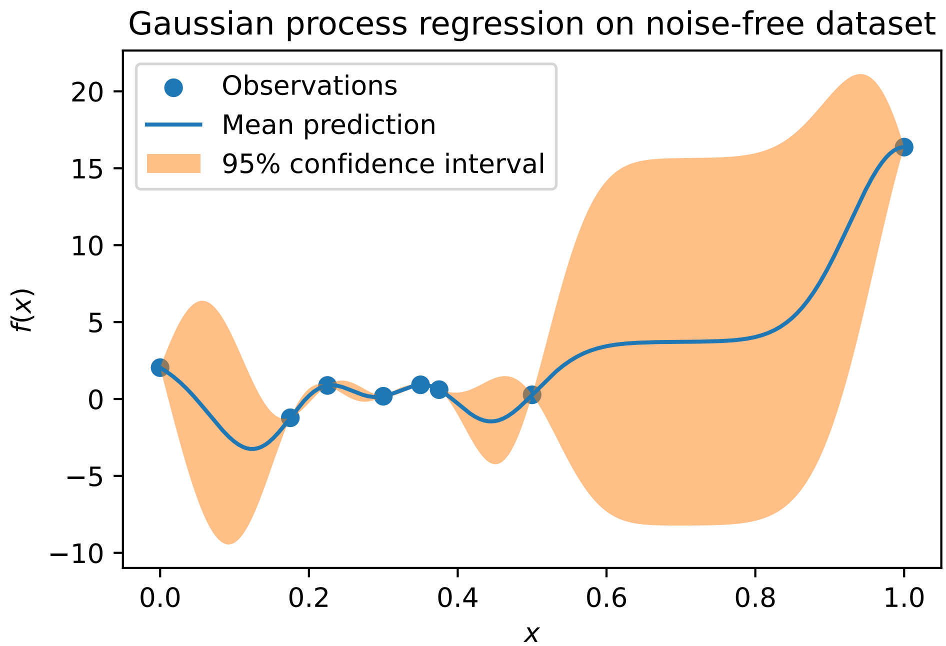

```
S.log
```

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

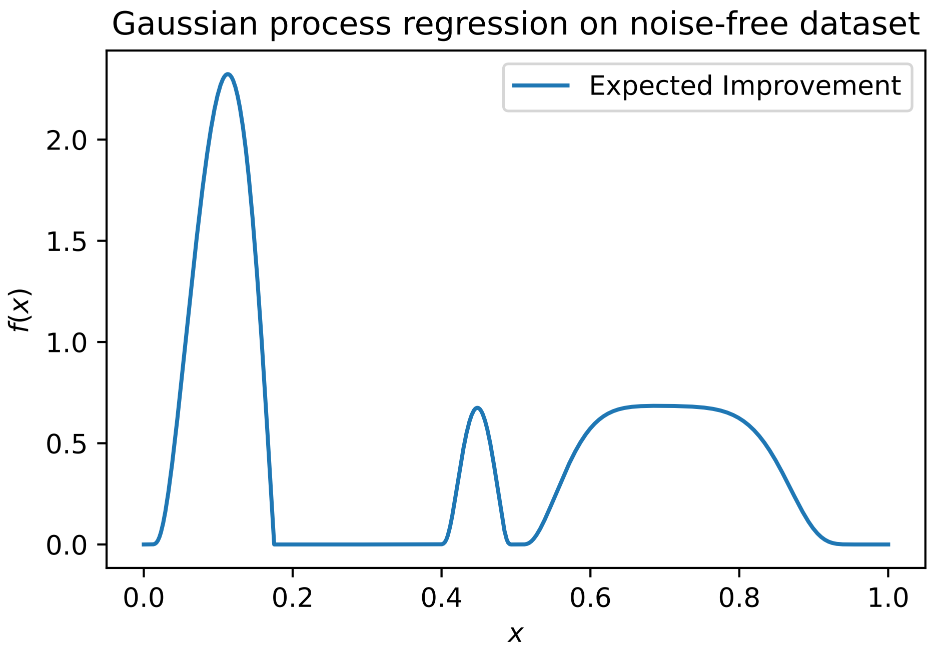
7.10 EI: The Forrester Example

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

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

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

S = Kriging(name='kriging',  seed=123, log_level=50, n_theta=1, noise=False, cod_type="nor
S.fit(X_train, y_train)

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

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




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

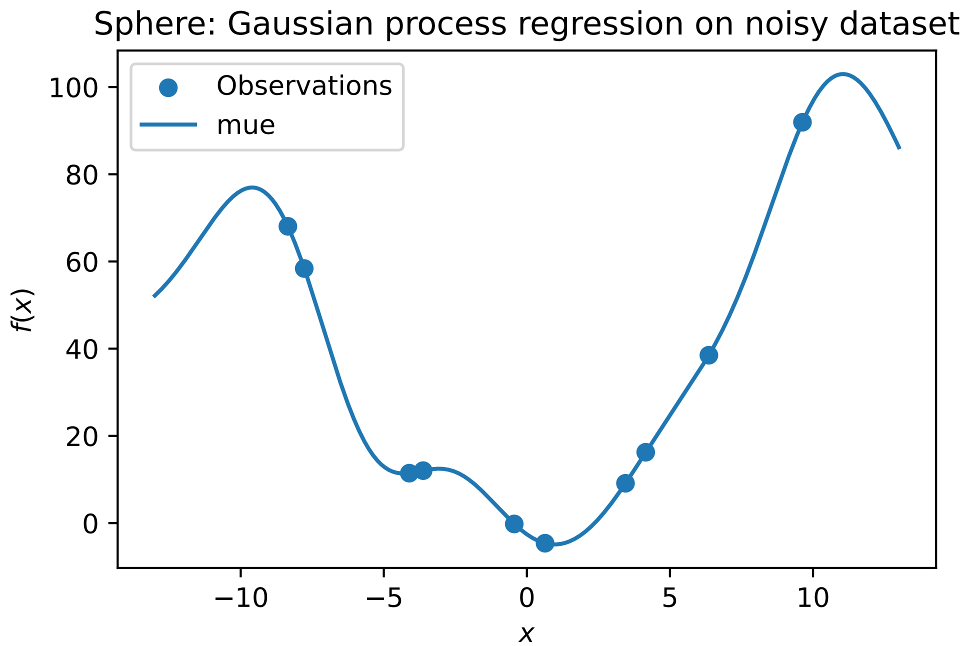


7.11 Noise

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

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

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

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

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

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

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

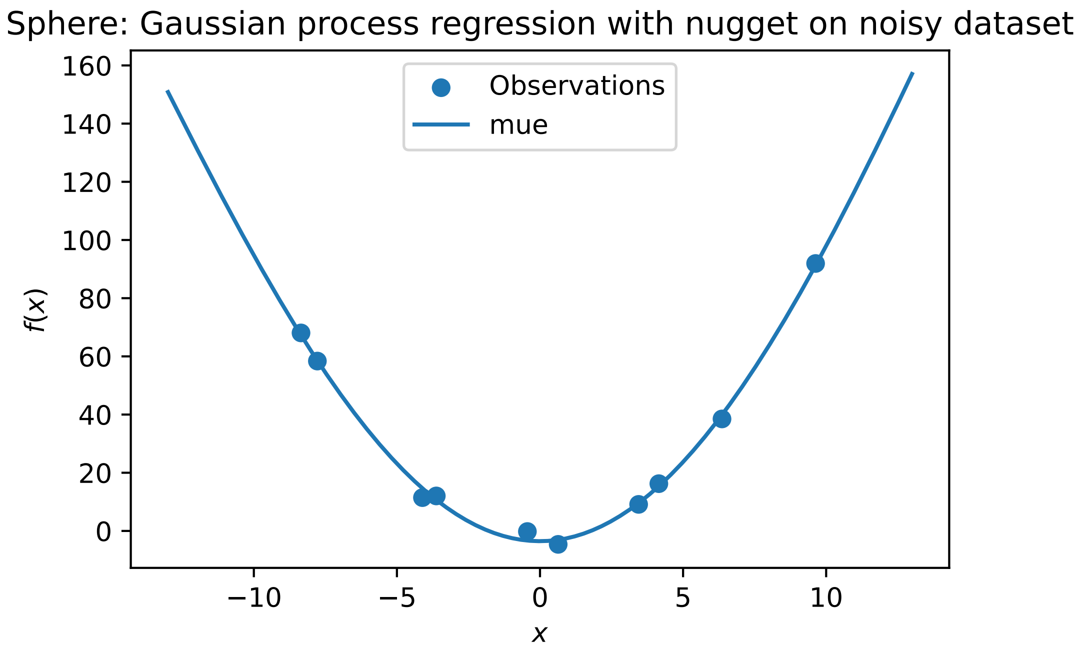



```
S.log
```

```
{'negLnLike': array([24.69806131]),
 'theta': array([1.31023943]),
 'p': array([2.]),
 'Lambda': array([None], dtype=object)}
```

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

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

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

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




```
S.log
```

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

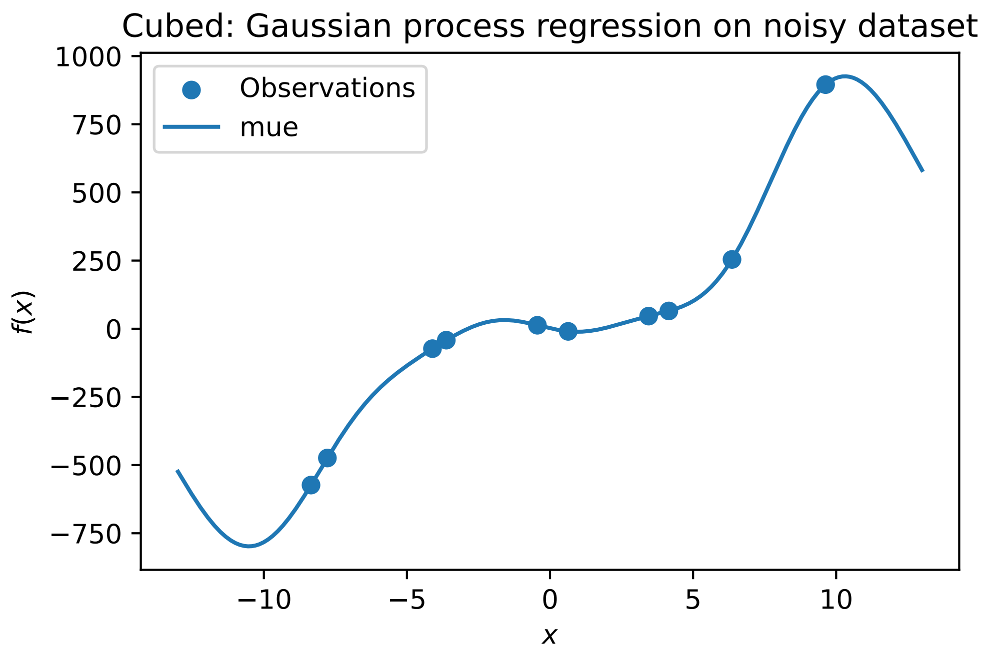
7.12 Cubic Function

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

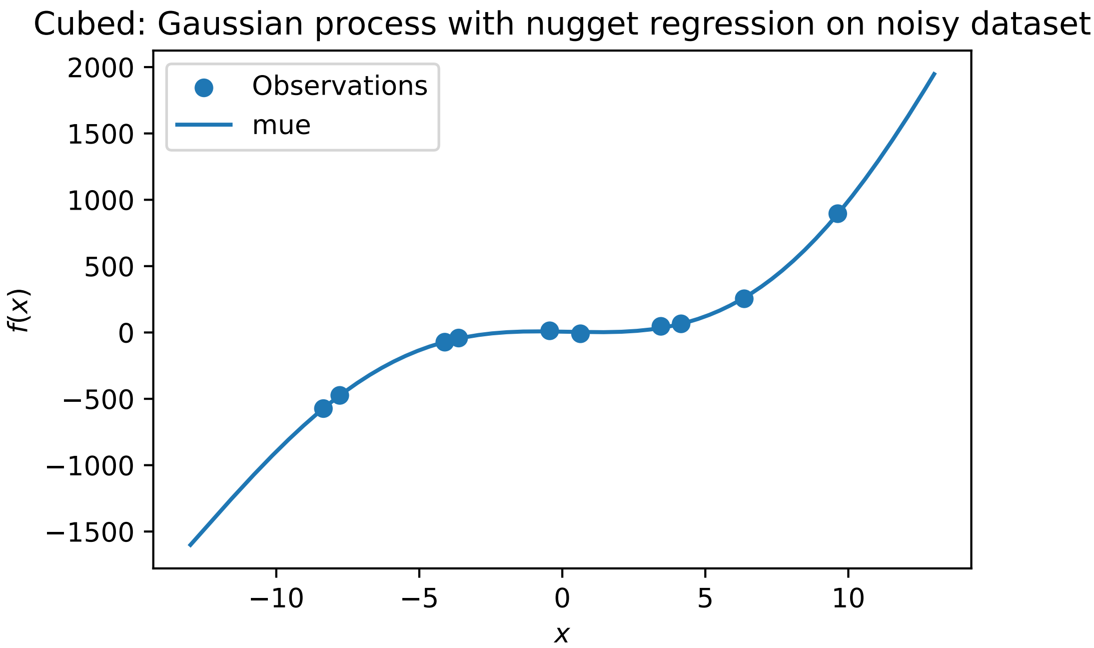
```python
from spotPython.build.kriging import Kriging
import matplotlib.pyplot as plt

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

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

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

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

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

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

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



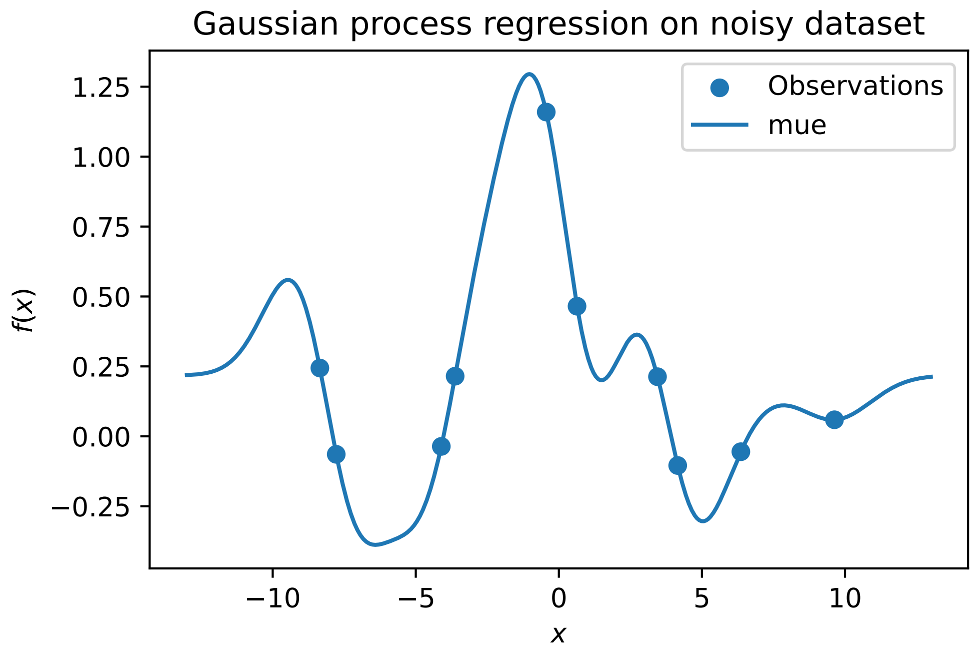

```python
S = Kriging(name='kriging',  seed=123, log_level=0, n_theta=1, noise=True)
S.fit(X_train, y_train)

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

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



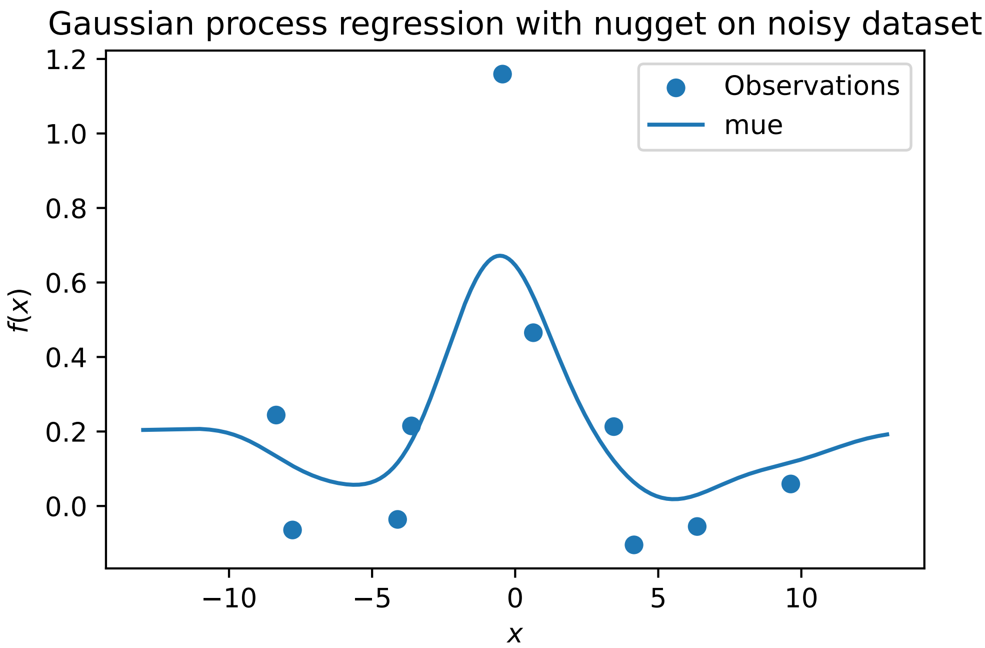

```
import numpy as np
import spotPython
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from spotPython.design.spacefilling import spacefilling
from spotPython.build.kriging import Kriging
import matplotlib.pyplot as plt

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

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

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

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

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

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

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




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

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

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



7.13 Factors

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

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

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

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

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

210.95481017555812

```
# vars(S)
```

```
# vars(Sf)
```

8 Hyperparameter Tuning and Noise

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

8.1 Example: `Spot` and the Noisy Sphere Function

```python
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
```

```python
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_bartz09_2023-06-27_02-22-17
```

8.1.1 The Objective Function: Noisy Sphere

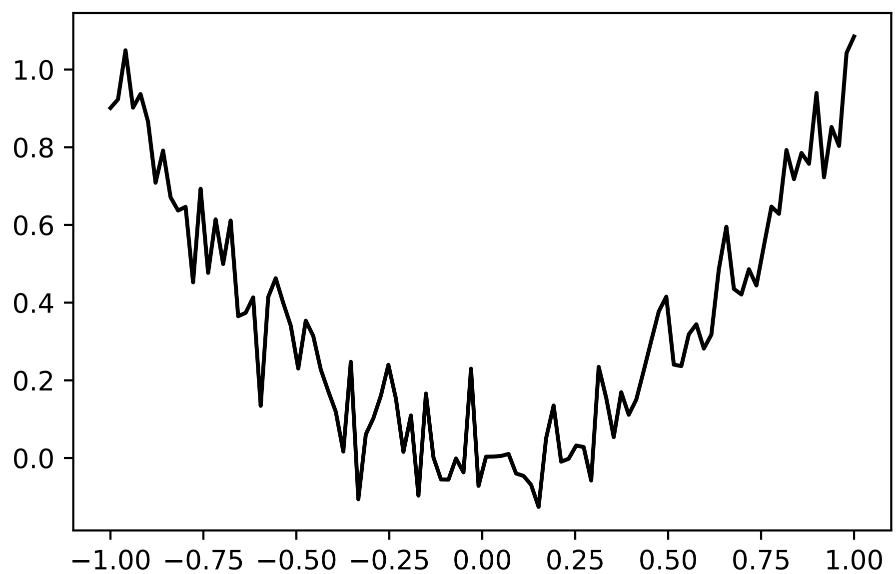
- The `spotPython` package provides several classes of objective functions.

- We will use an analytical objective function with noise, i.e., a function that can be described by a (closed) formula:
$$f(x) = x^2 + \epsilon$$
- Since `sigma` is set to `0.1`, noise is added to the function:

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

- A plot illustrates the noise:

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

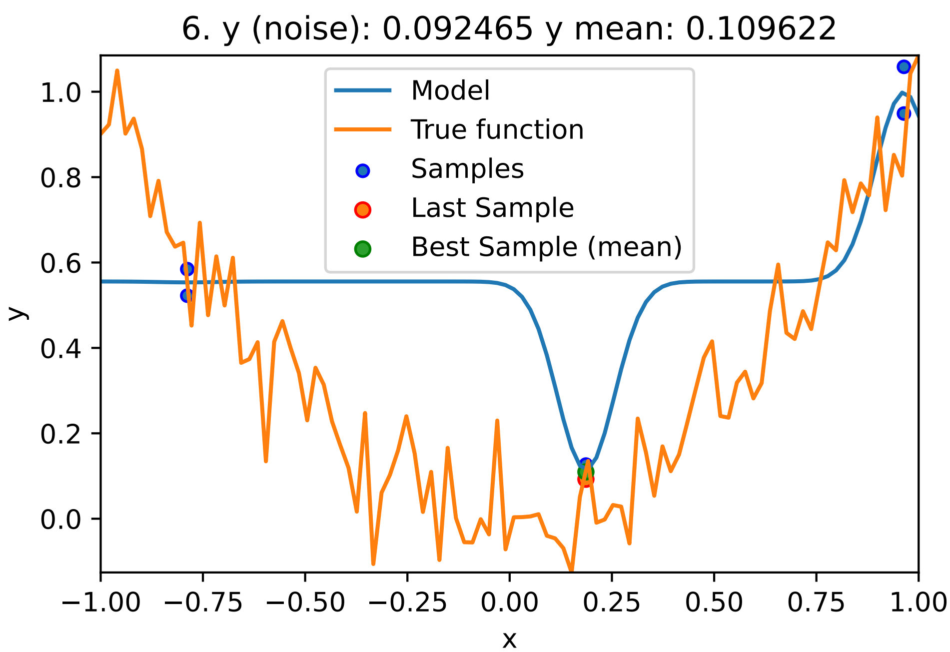


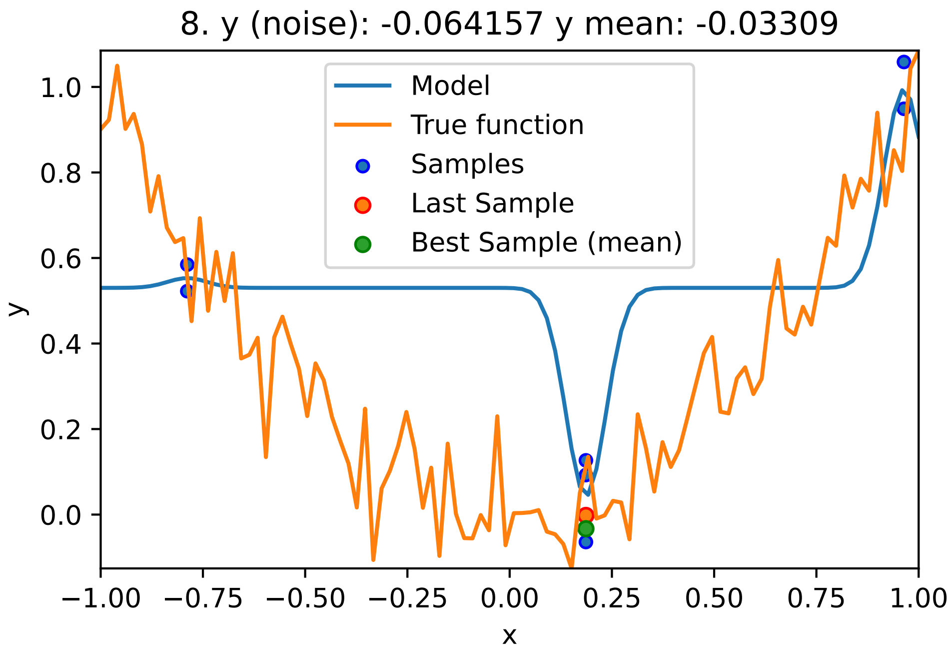
`Spot` is adopted as follows to cope with noisy functions:

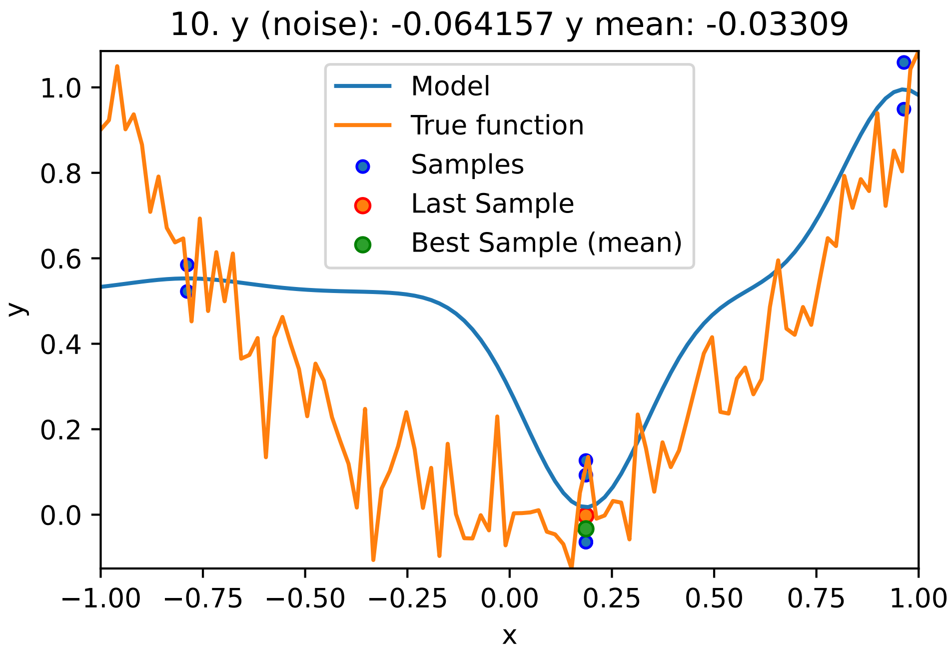
1. `fun_repeats` is set to a value larger than 1 (here: 2)
2. `noise` is set to `true`. Therefore, a nugget (`Lambda`) term is added to the correlation matrix
3. `init size` (of the `design_control` dictionary) is set to a value larger than 1 (here: 2)

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

```
spot_1_noisy.run()
```








```
<spotPython.spot.spot.Spot at 0x1515245b0>
```

8.2 Print the Results

```
spot_1_noisy.print_results()
```

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

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

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

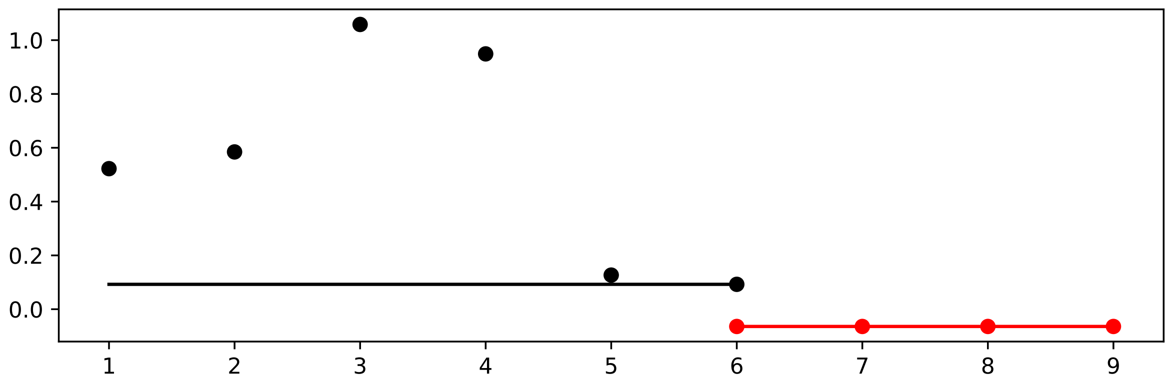


Figure 8.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

8.3 Noise and Surrogates: The Nugget Effect

8.3.1 The Noisy Sphere

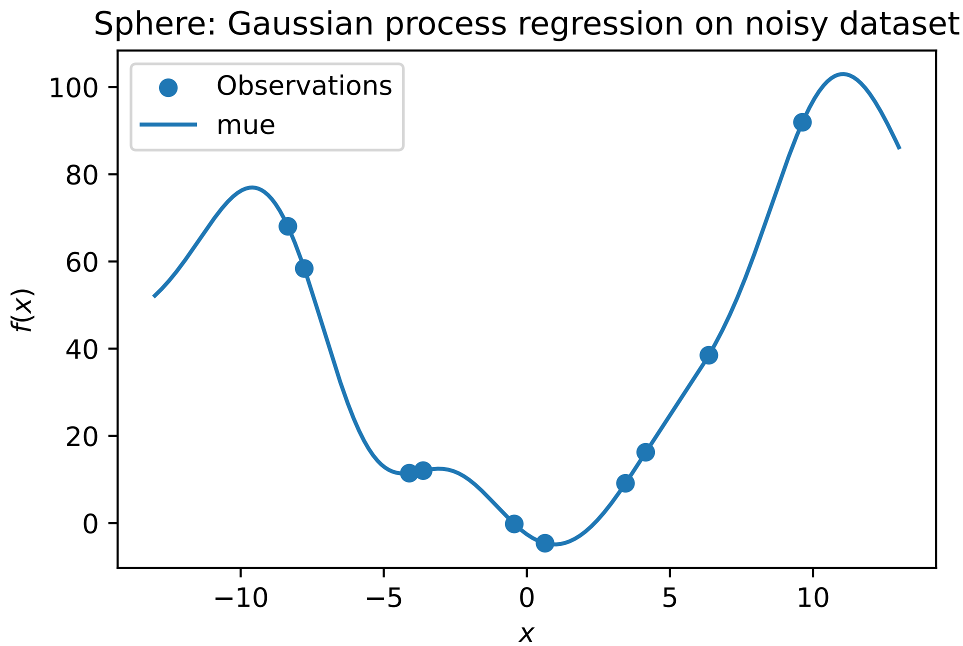
8.3.1.1 The Data

- We prepare some data first:

```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
```

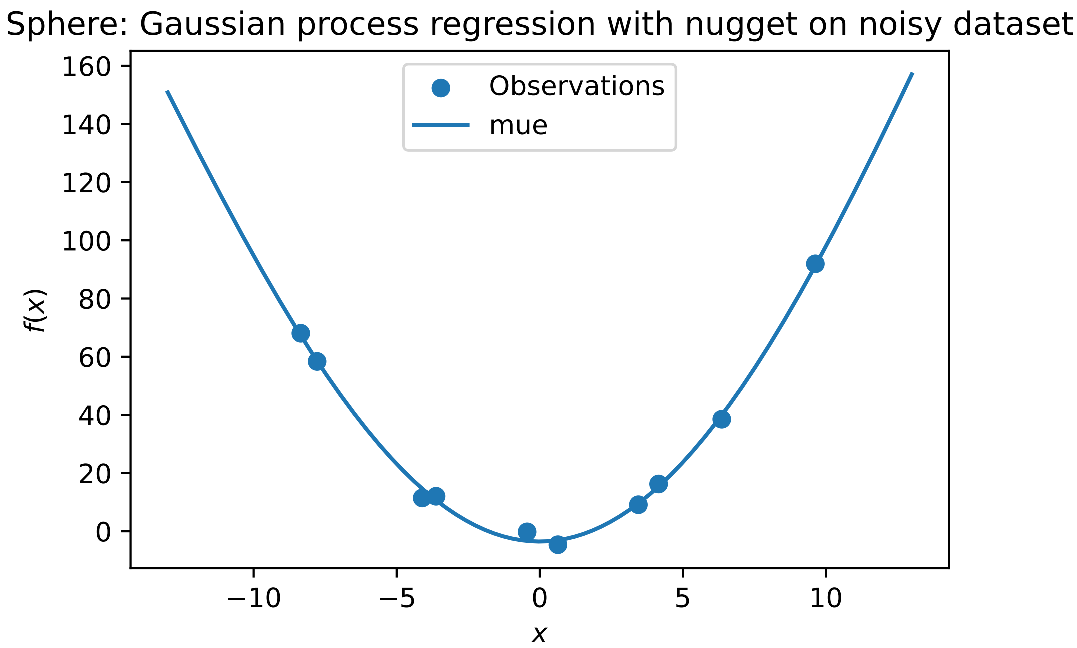
- A surrogate without nugget is fitted to these data:

```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.

8.4 Exercises

8.4.1 Noisy `fun_cubed`

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

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

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

8.4.2 fun_runge

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

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

8.4.3 fun_forrester

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

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

8.4.4 fun_xsin

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

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

9 Handling Noise: Optimal Computational Budget Allocation in Spot

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

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

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

9.1.1 The Objective Function: Noisy Sphere

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

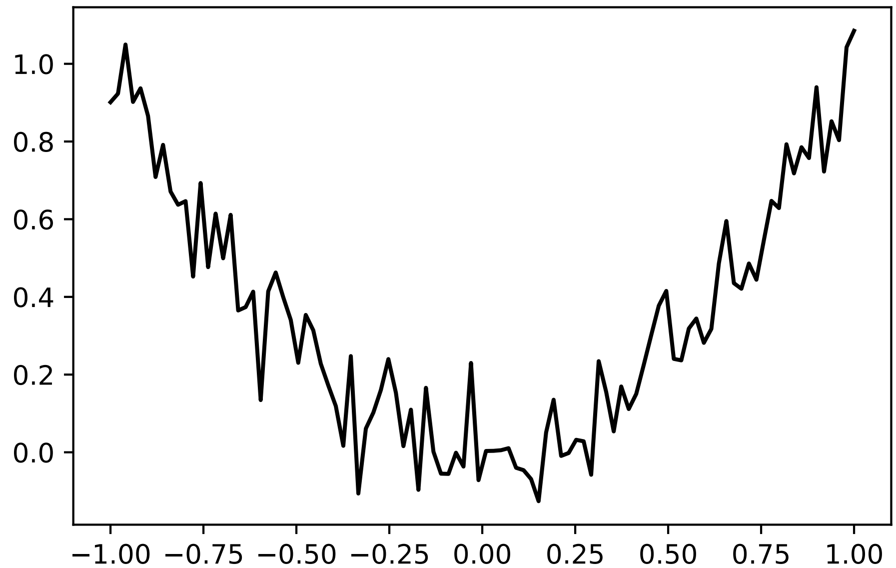
$$f(x) = x^2 + \epsilon$$

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

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

A plot illustrates the noise:

```python
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()
```



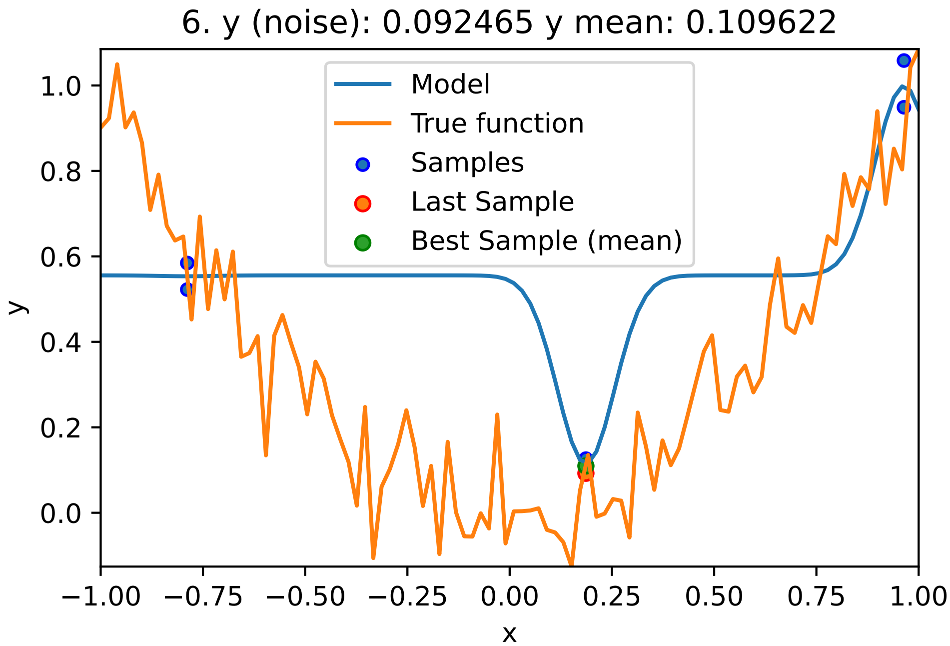
`Spot` is adopted as follows to cope with noisy functions:

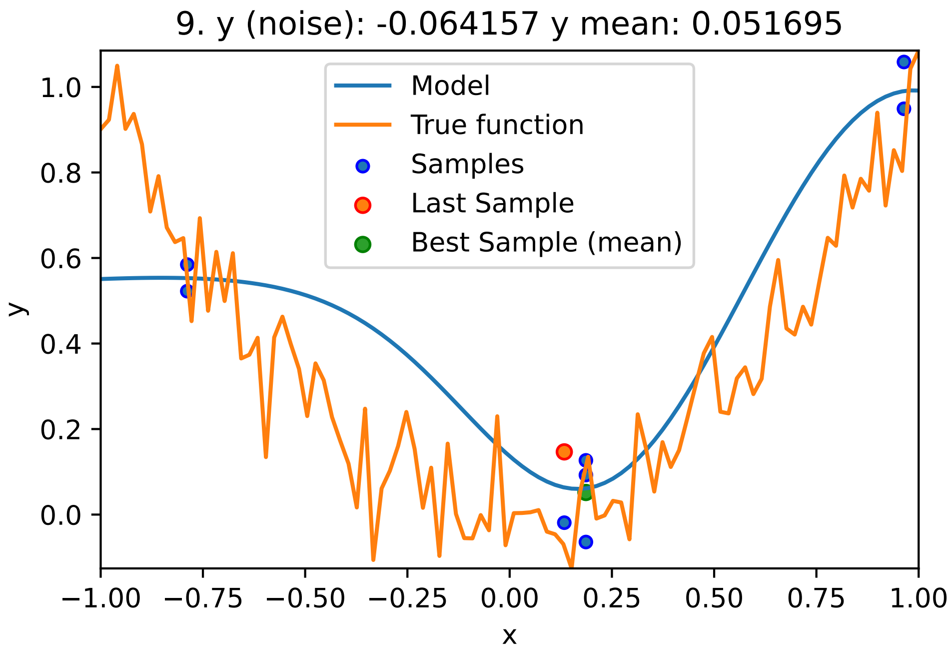
1. `fun_repeats` is set to a value larger than 1 (here: 2)
2. `noise` is set to `true`. Therefore, a nugget (`Lambda`) term is added to the correlation matrix
3. `init size` (of the `design_control` dictionary) is set to a value larger than 1 (here: 2)

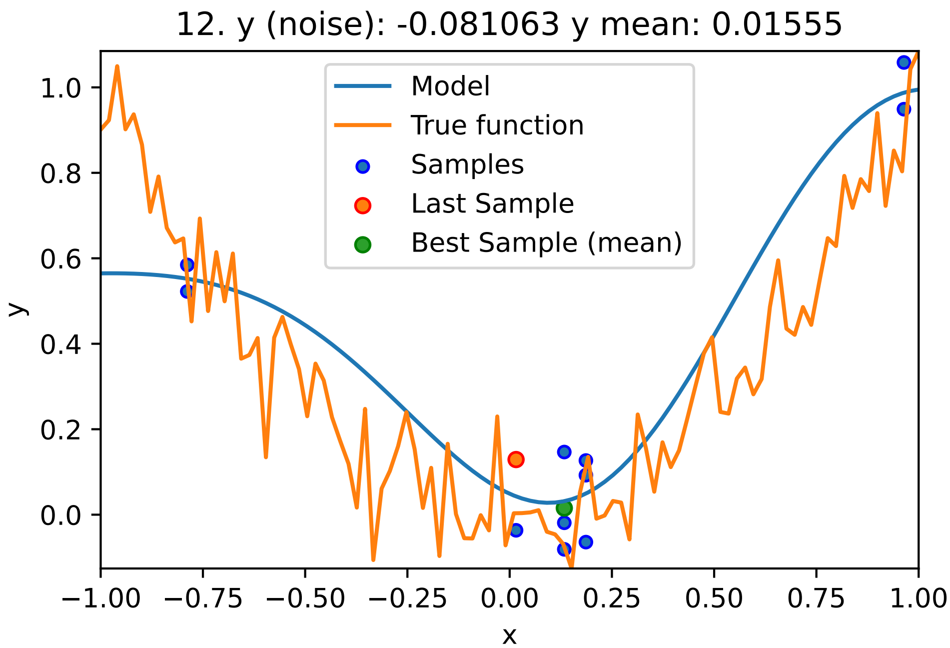
```python
spot_1_noisy = spot.Spot(fun=fun,
                   lower = np.array([-1]),
                   upper = np.array([1]),
                   fun_evals = 50,
                   fun_repeats = 2,
                   infill_criterion="ei",
                   noise = True,
                   tolerance_x=0.0,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

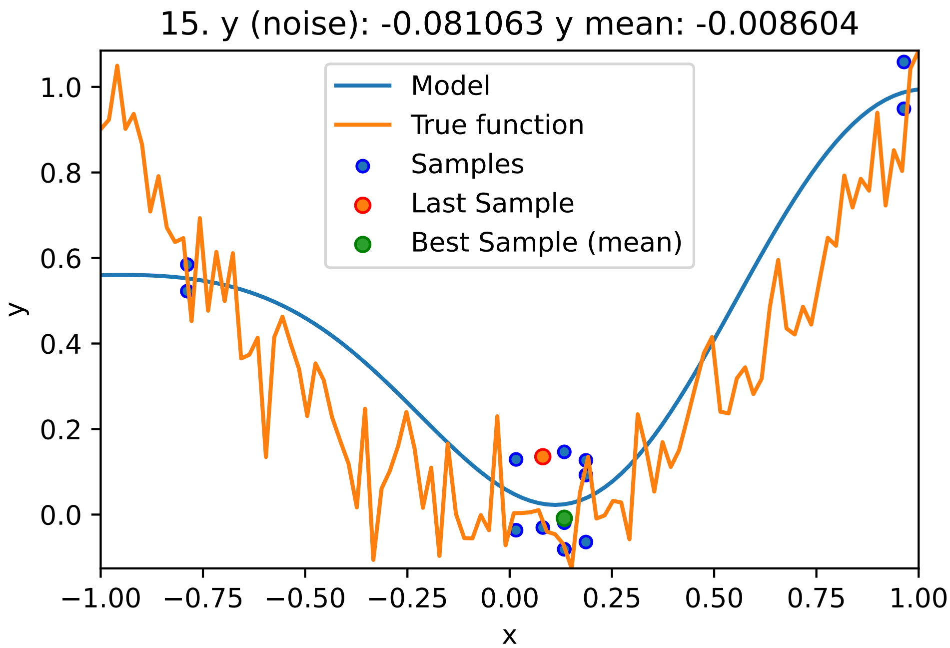
```
                     show_models=True,
                     fun_control = fun_control,
                     design_control={"init_size": 3,
                                     "repeats": 2},
                     surrogate_control={"noise": True})
```

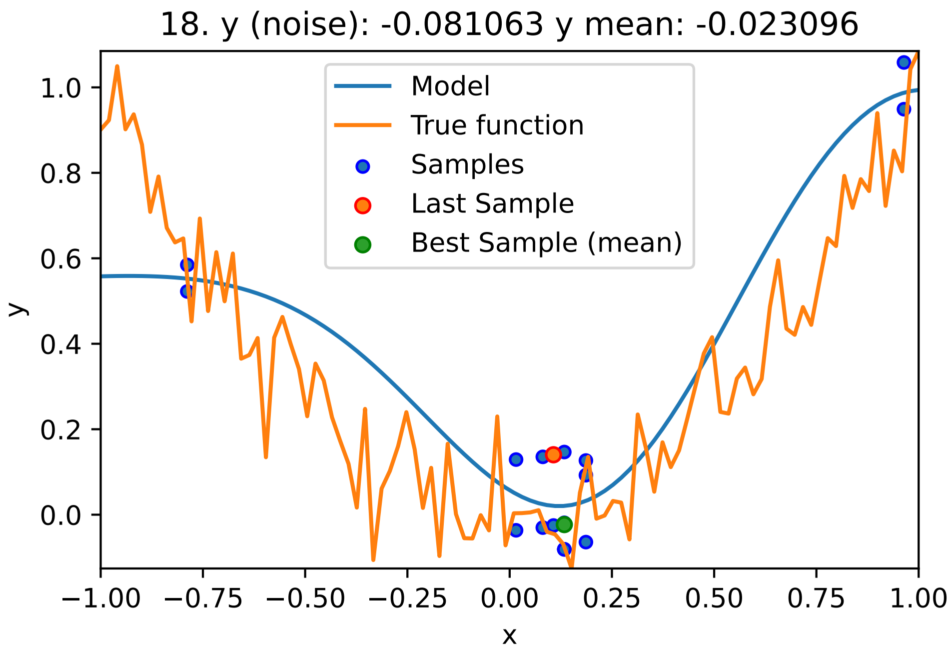
```
spot_1_noisy.run()
```

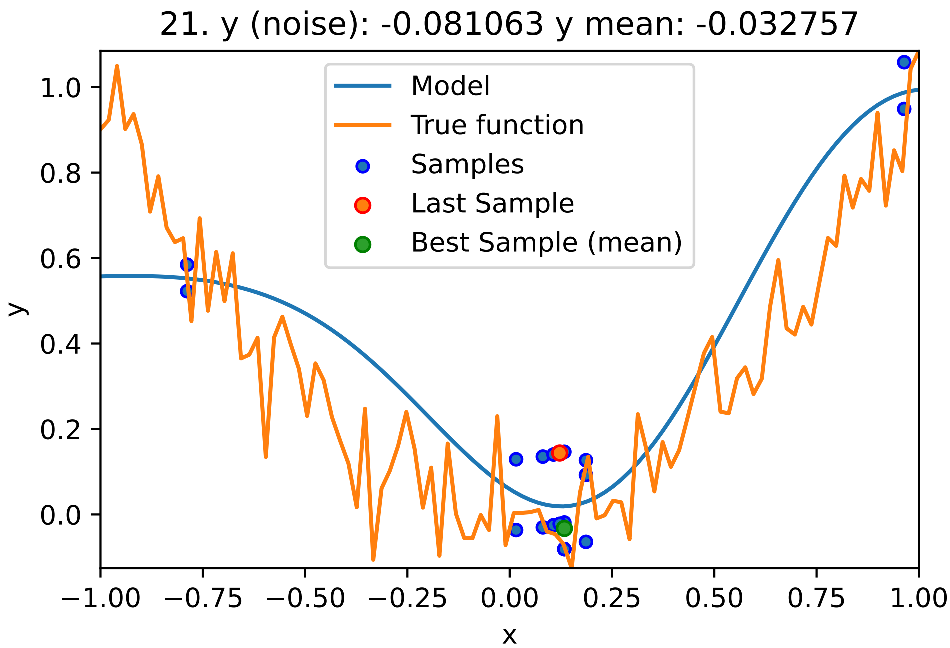


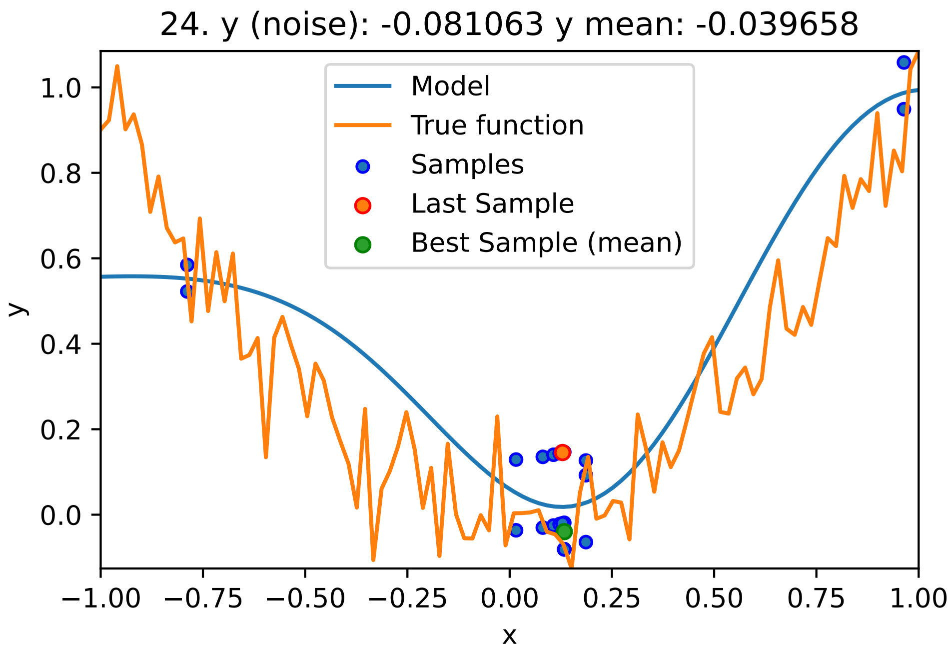

9. y (noise): -0.064157 y mean: 0.051695
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x

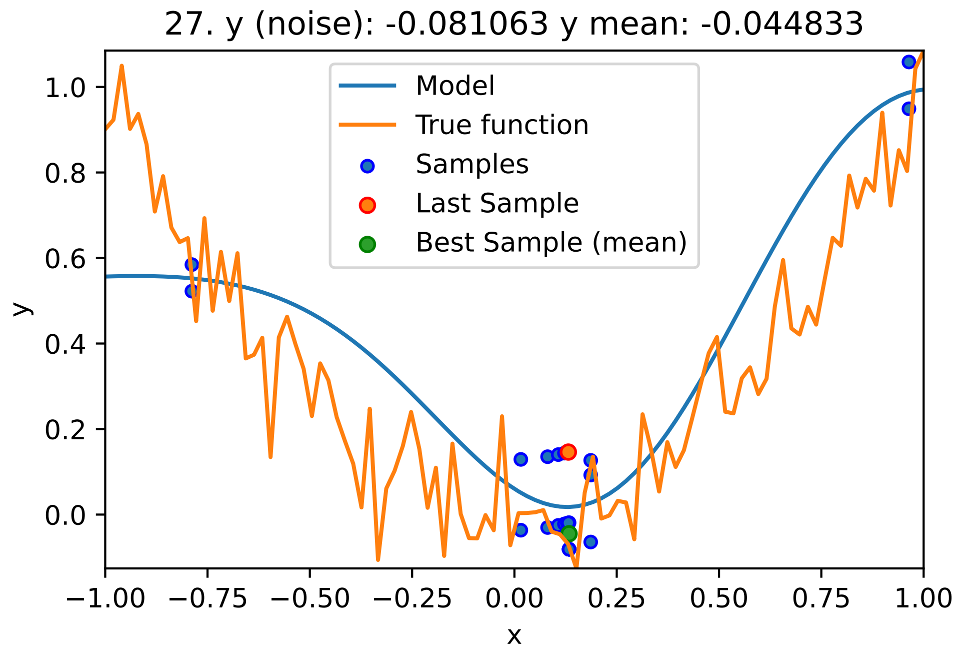

12. y (noise): -0.081063 y mean: 0.01555
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x

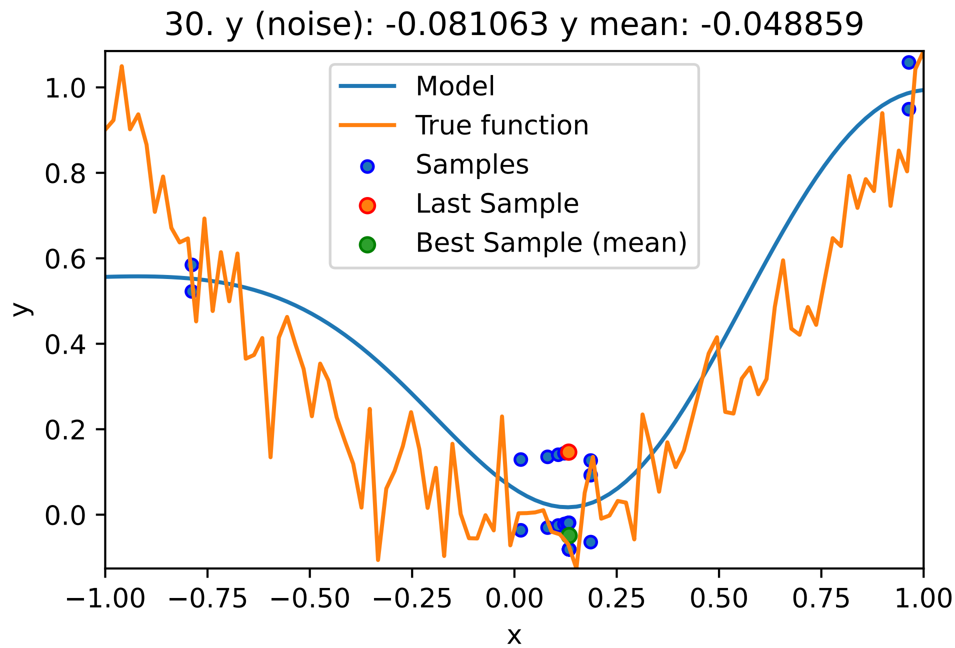

15. y (noise): -0.081063 y mean: -0.008604
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

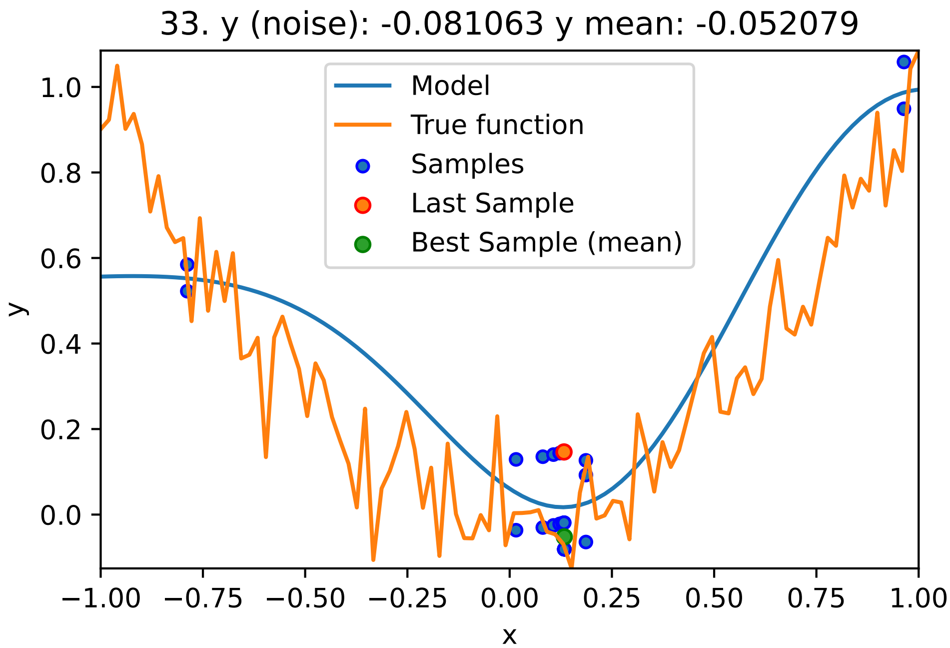

18. y (noise): -0.081063 y mean: -0.023096
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

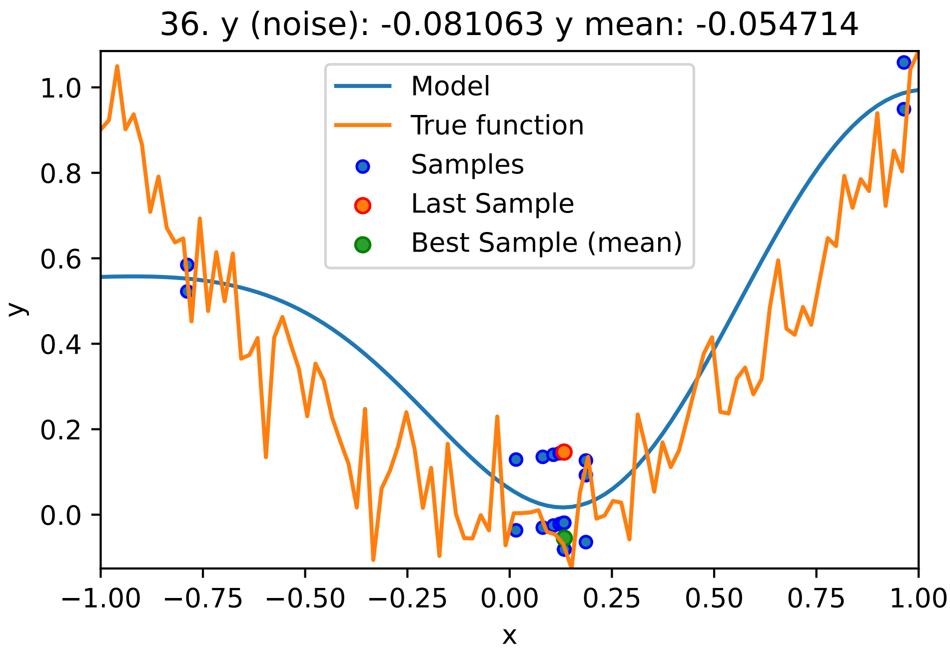

21. y (noise): -0.081063 y mean: -0.032757
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

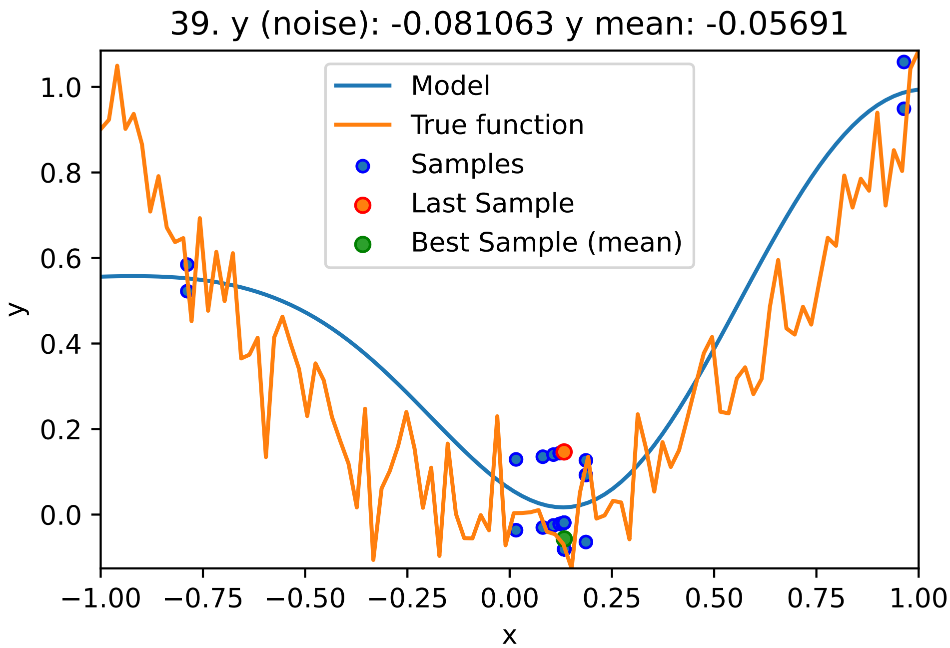

24. y (noise): -0.081063 y mean: -0.039658
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

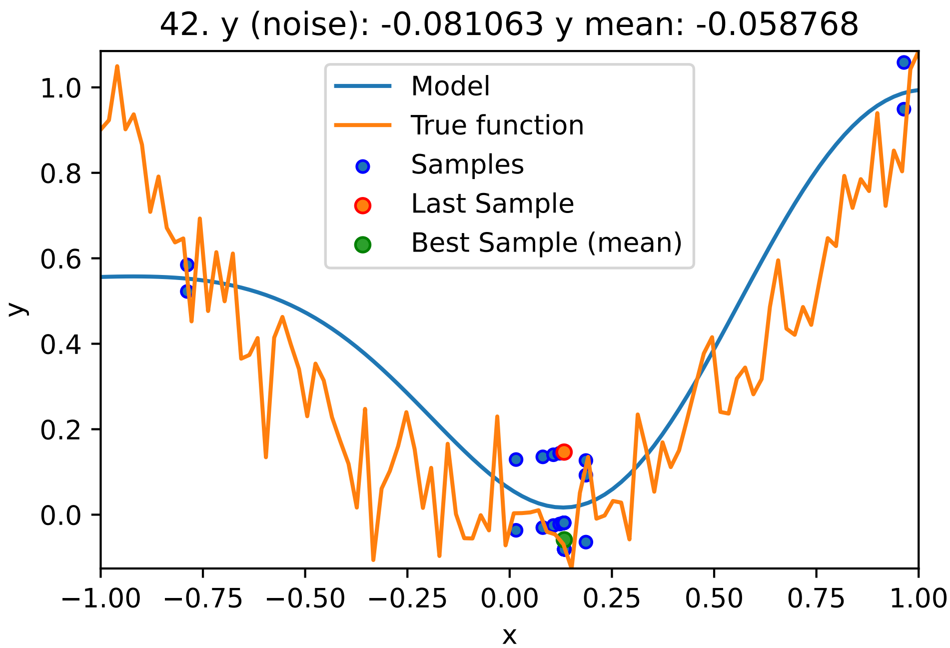

27. y (noise): -0.081063 y mean: -0.044833
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

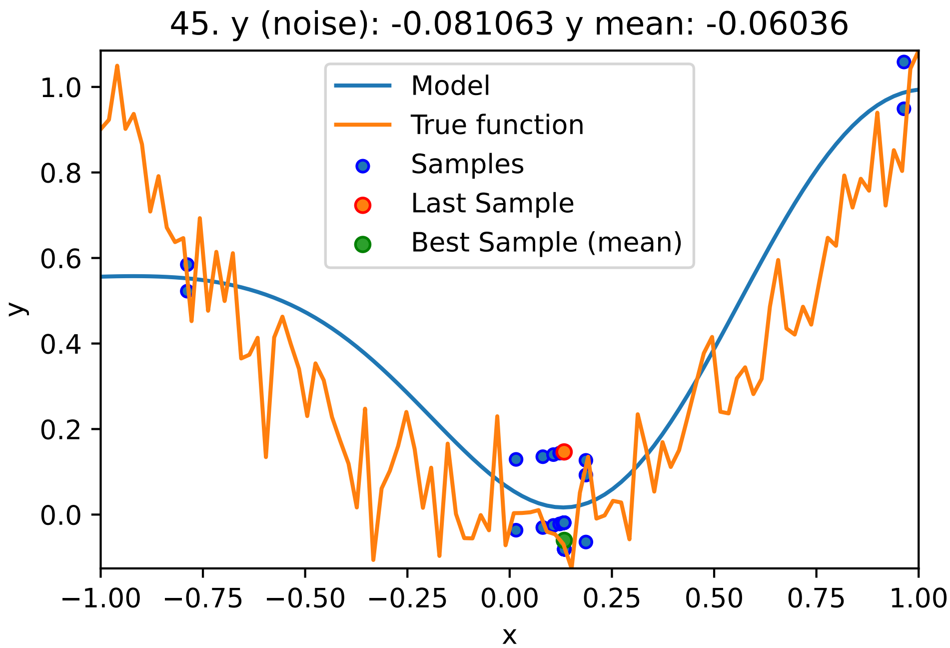

30. y (noise): -0.081063 y mean: -0.048859
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


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

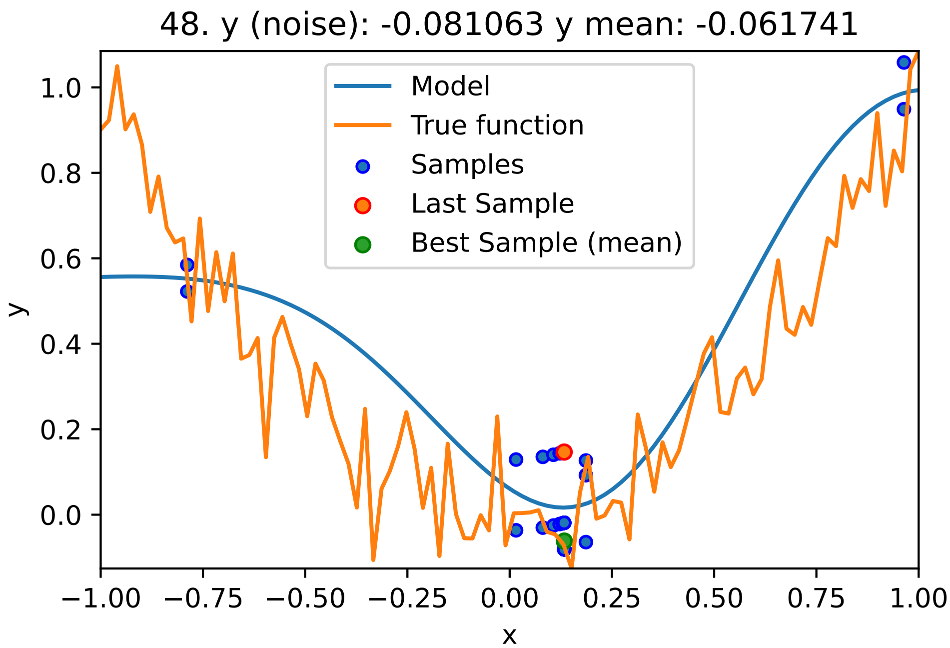

36. y (noise): -0.081063 y mean: -0.054714
Model
True function
Samples
Last Sample
Best Sample (mean)
1.0
0.8
0.6
0.4
0.2
0.0
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x


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

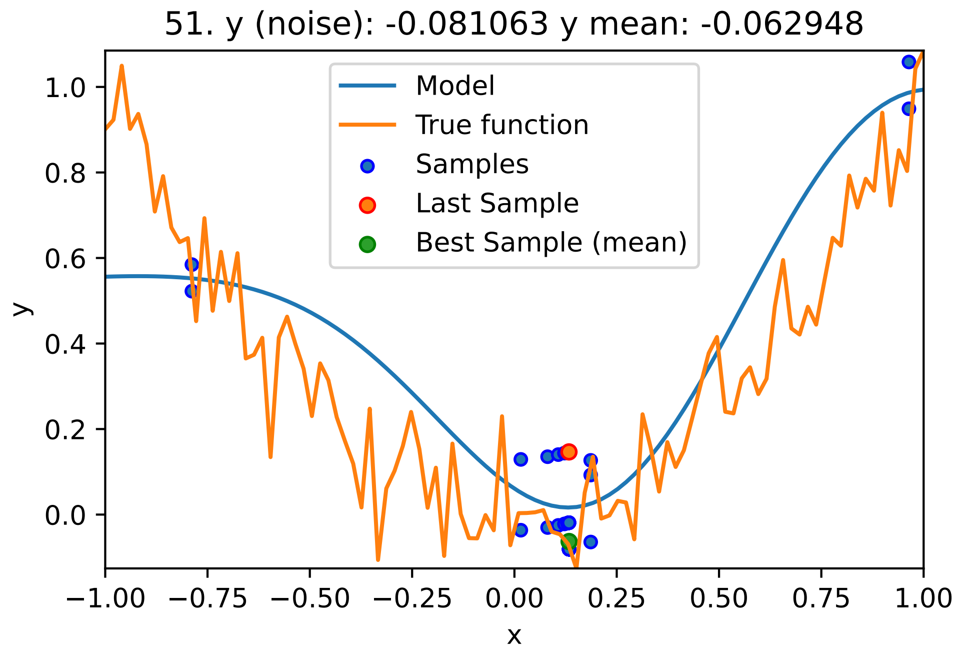

42. y (noise): -0.081063 y mean: -0.058768
Model
True function
Samples
Last Sample
Best Sample (mean)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00



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



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




```
<spotPython.spot.spot.Spot at 0x1660245b0>
```

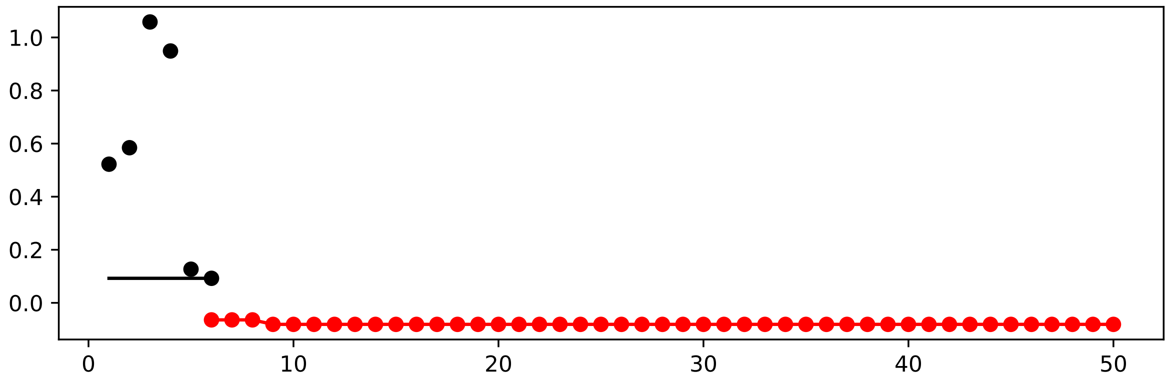
9.2 Print the Results

```
spot_1_noisy.print_results()
```

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

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

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



9.3 Noise and Surrogates: The Nugget Effect

9.3.1 The Noisy Sphere

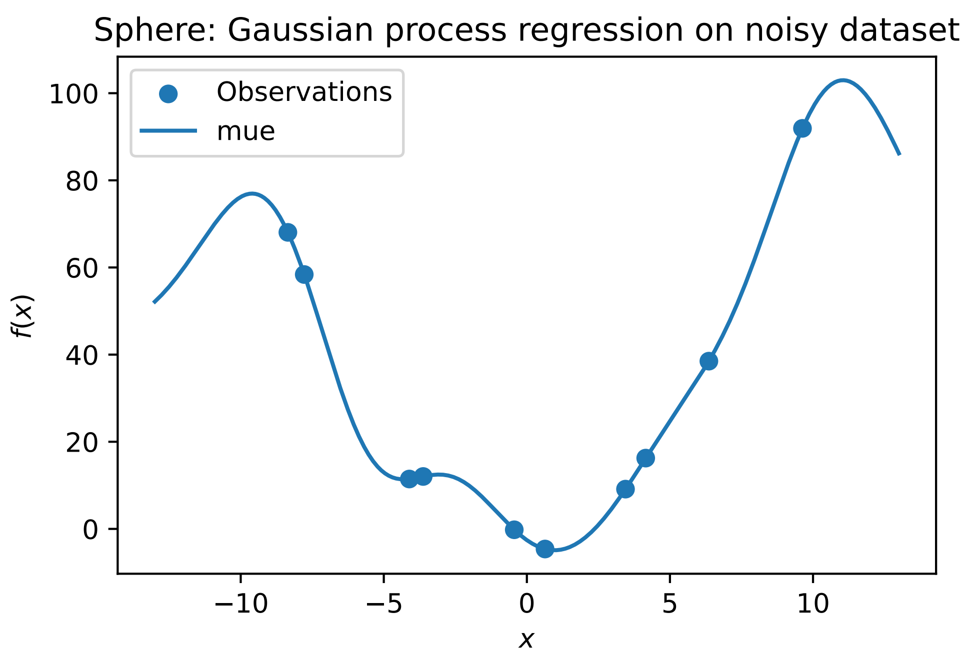
9.3.1.1 The Data

We prepare some data first:

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

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

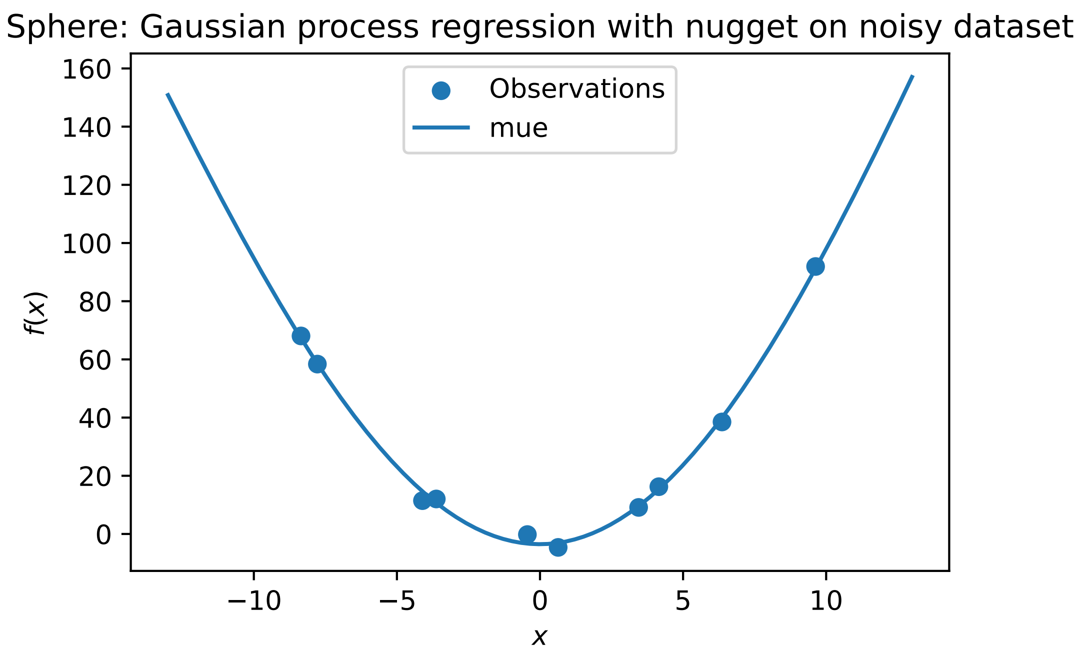
A surrogate without nugget is fitted to these data:

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

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

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



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

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



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

```
S.Lambda
```

```
S_nug.Lambda
```

```
9.088150066416743e-05
```

We see:

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

9.4 Exercises

9.4.1 Noisy `fun_cubed`

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

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

9.4.2 `fun_runge`

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

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

9.4.3 `fun_forrester`

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

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

9.4.4 `fun_xsin`

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

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

```
spot_1_noisy.mean_y.shape[0]
```

```
18
```

10 HPT: sklearn SVC on Moons Data

This document refers to the following software versions:

- `python`: 3.10.10

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

```
spotPython                    0.2.46
spotRiver                     0.0.93
```

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

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

```
!pip install spotPython
```

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

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

10.1 Step 1: Setup

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

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

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

```
10-sklearn_bartz09_1min_5init_2023-06-27_02-23-55
```

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

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

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

10.3 Step 3: SKlearn Load Data (Classification)

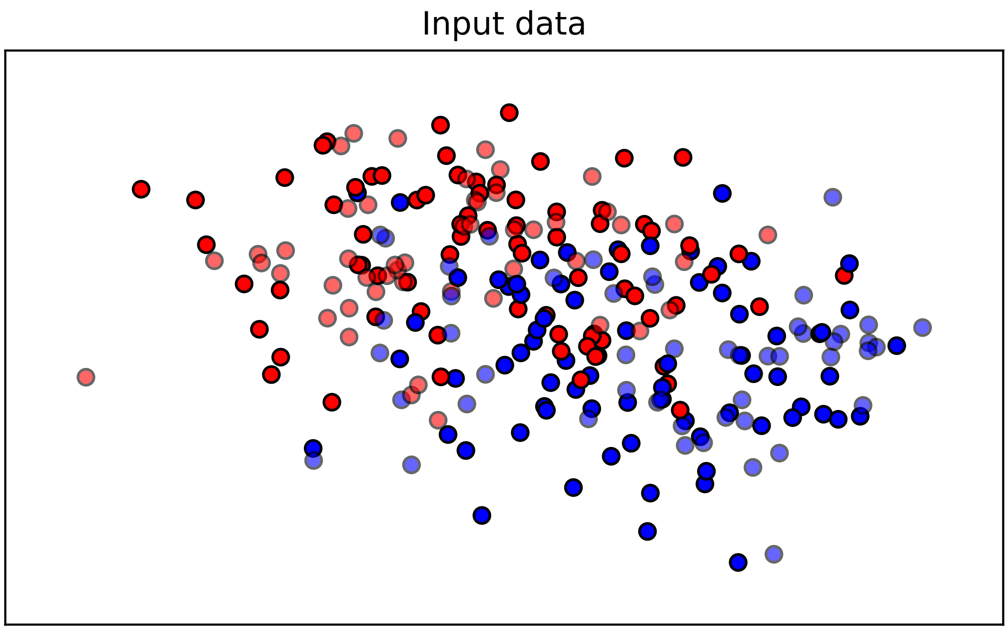
Randomly generate classification data.

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

```python
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

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

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




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

10.4 Step 4: Specification of the Preprocessing Model

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

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

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

```python
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:

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

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

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

```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
```

```python
# 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.

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


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

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

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

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

10.6.1 Modify hyperparameter of type numeric and integer (boolean)

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

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

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

10.6.2 Modify hyperparameter of type factor

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

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

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

10.6.3 Optimizers

Optimizers are described in Section 14.6.1.

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

There are two metrics:

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

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

10.7.1 Predict Classes or Class Probabilities

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

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

10.8 Step 8: Calling the SPOT Function

10.8.1 Preparing the SPOT Call

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

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

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

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

10.8.2 The Objective Function

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

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

10.8.3 Run the `Spot` Optimizer

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

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

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

10.8.4 Starting the Hyperparameter Tuning

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

```
spotPython tuning: 5.691103166702708 [----------] 2.81%

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

spotPython tuning: 5.691103166702708 [#---------] 6.24%

spotPython tuning: 5.691103166702708 [#---------] 7.72%
```

```
spotPython tuning: 5.691103166702708 [#---------] 9.17%

spotPython tuning: 5.691103166702708 [#---------] 11.54%

spotPython tuning: 5.691103166702708 [#---------] 13.82%

spotPython tuning: 5.691103166702708 [##--------] 16.04%

spotPython tuning: 5.691103166702708 [##--------] 18.39%

spotPython tuning: 5.691103166702708 [##--------] 20.68%

spotPython tuning: 5.691103166702708 [##--------] 23.20%

spotPython tuning: 5.691103166702708 [###-------] 25.78%

spotPython tuning: 5.691103166702708 [###-------] 34.17%

spotPython tuning: 5.691103166702708 [#####-----] 46.08%

spotPython tuning: 5.691103166702708 [######----] 58.85%

spotPython tuning: 5.691103166702708 [#######---] 71.58%

spotPython tuning: 5.691103166702708 [########--] 83.96%

spotPython tuning: 5.691103166702708 [##########] 96.49%

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

<spotPython.spot.spot.Spot at 0x17ce1fca0>
```

10.9 Step 9: Results

```
SAVE = False
LOAD = False

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

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

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

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

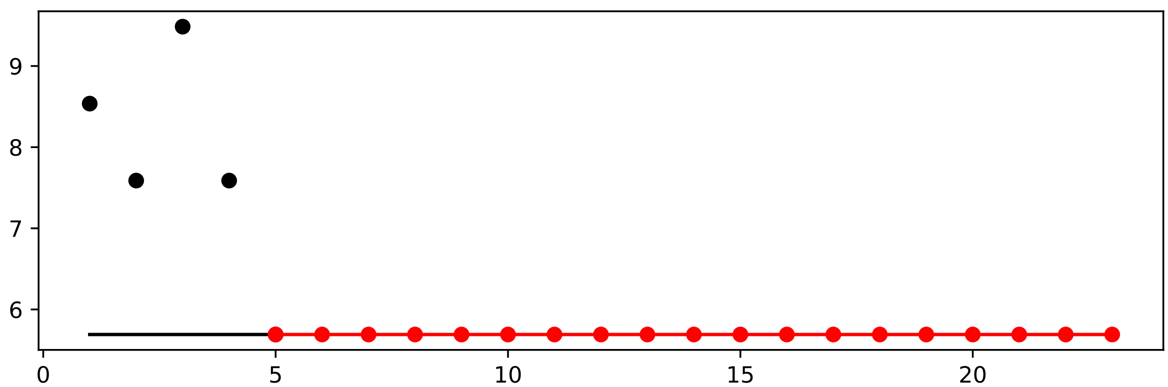


Figure 10.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- Print the results

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

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

```
| C            | float  | 1.0        |     0.1 |    10.0 |   3.6280771109650245 | None
| kernel       | factor | rbf        |     0.0 |     2.0 |                  1.0 | None
| degree       | int    | 3          |     3.0 |     3.0 |                  3.0 | None
| gamma        | factor | scale      |     0.0 |     1.0 |                  0.0 | None
| coef0        | float  | 0.0        |     0.0 |     0.0 |                  0.0 | None
| shrinking    | factor | 0          |     0.0 |     1.0 |                  1.0 | None
| probability  | factor | 0          |     0.0 |     1.0 |                  0.0 | None
| tol          | float  | 0.001      |   0.001 |    0.01 | 0.006642600916881275 | None
| cache_size   | float  | 200.0      |   100.0 |   400.0 |   202.03372626175258 | None
| break_ties   | factor | 0          |     0.0 |     1.0 |                  1.0 | None
```

10.9.1 Show variable importance

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

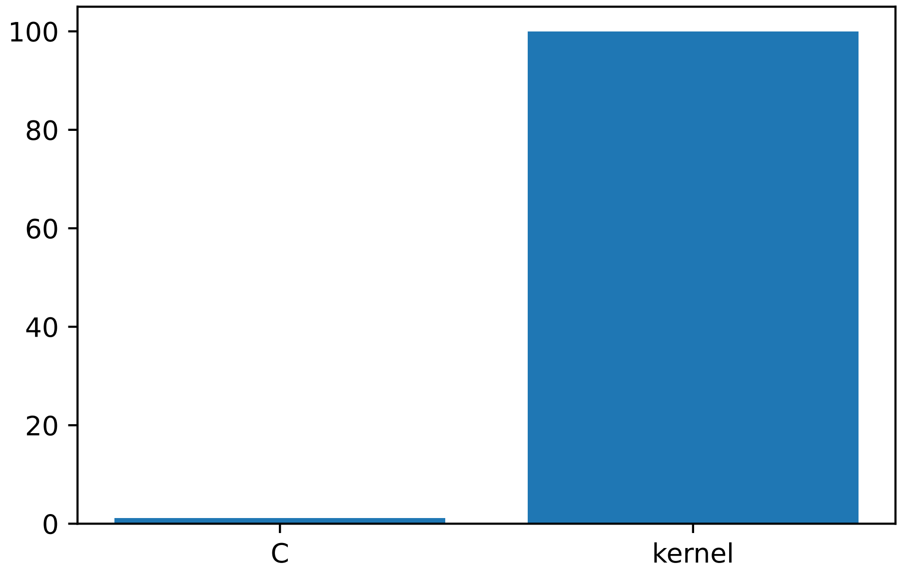


Figure 10.2: Variable importance plot, threshold 0.025.

10.9.2 Get Default Hyperparameters

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

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

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

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

10.9.3 Get SPOT Results

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

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

```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)
```

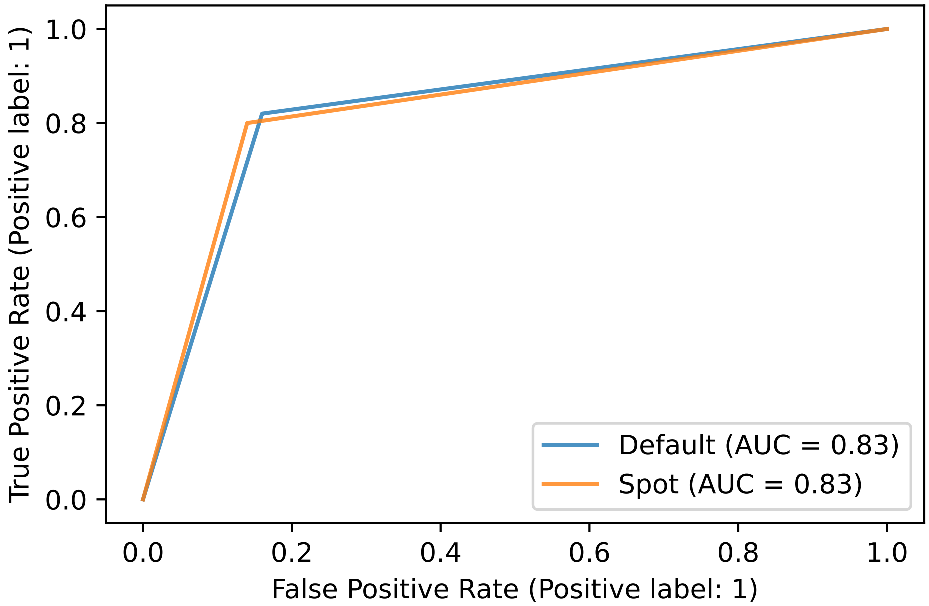
```
[{'C': 3.6280771109650245,
  'kernel': 'poly',
  'degree': 3,
  'gamma': 'scale',
  'coef0': 0.0,
  'shrinking': 1,
  'probability': 0,
  'tol': 0.006642600916881275,
  'cache_size': 202.03372626175258,
  'break_ties': 1}]
```

```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
```

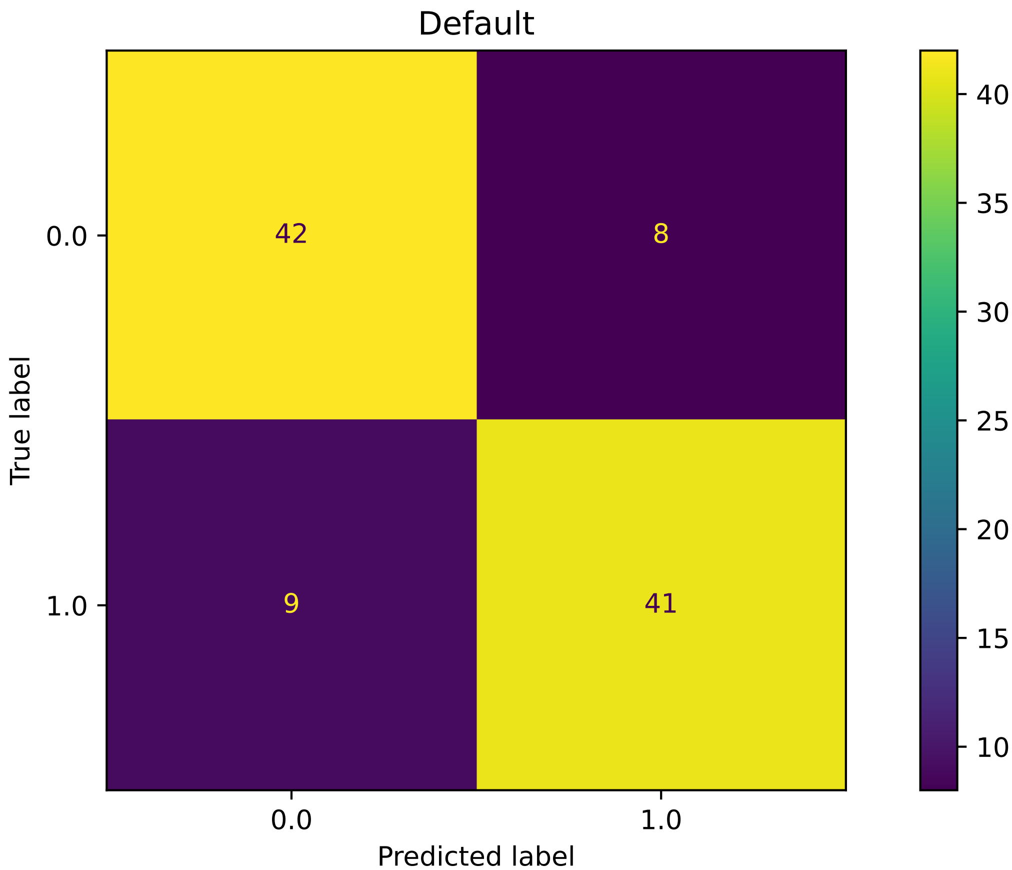
```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('svc',
                 SVC(C=3.6280771109650245, break_ties=1,
                     cache_size=202.03372626175258, kernel='poly',
                     probability=0, shrinking=1, tol=0.006642600916881275))])
```

10.9.4 Plot: Compare Predictions

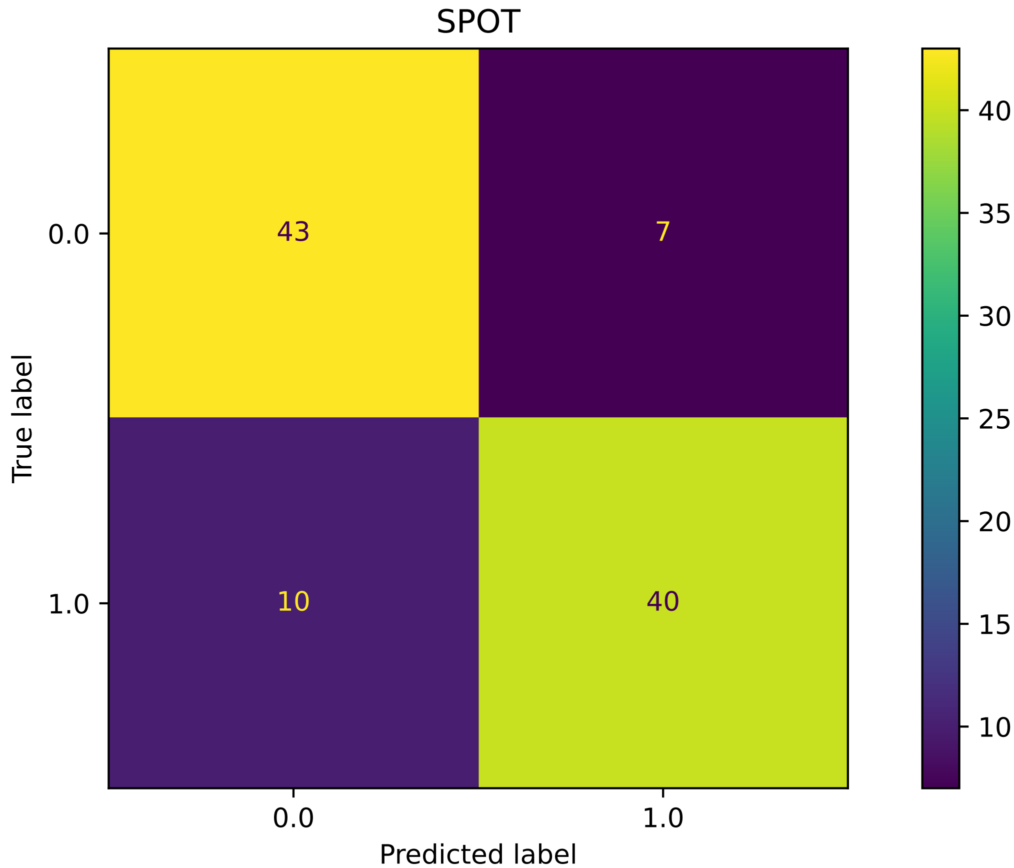
```python
from spotPython.plot.validation import plot_roc
plot_roc([model_default, model_spot], fun_control, model_names=["Default", "Spot"])
```




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




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



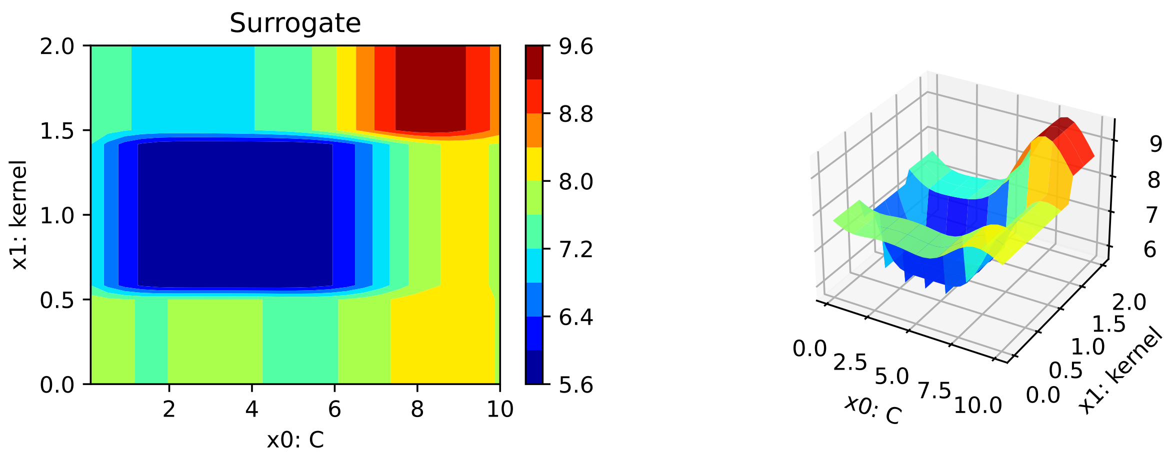

```
min(spot_tuner.y), max(spot_tuner.y)
```

```
(5.691103166702708, 9.485171944504513)
```

10.9.5 Detailed Hyperparameter Plots

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

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



10.9.6 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

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

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

10.9.7 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

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

11 HPT: PyTorch With fashionMNIST

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

This document refers to the following software versions:

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

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

```
spotPython                0.2.46
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
```

11.1 Step 1: Setup

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

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

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

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_bartz09_1min_5init_2023-06-27_02-31-02
```

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

`spotPython` uses a Python dictionary for storing the information required for the hyperparameter tuning process, which was described in Section 14.2.

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

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

11.3 Step 3: PyTorch Data Loading

11.3.1 Load fashionMNIST Data

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

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

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

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

11.4 Step 4: Specification of the Preprocessing Model

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

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

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

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


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

```
            k_folds=k_folds,
            patience=patience,
            optimizer=optimizer,
            sgd_momentum=sgd_momentum,
        )
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28 * 28, 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 14.5.1.

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

11.5.1 The Search Space

11.5.2 Configuring the Search Space With spotPython

11.5.2.1 The `hyper_dict` Hyperparameters for the Selected Algorithm

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

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

```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}}
```

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

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

11.6.1 Modify hyperparameter of type numeric and integer (boolean)

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

> Caution: Small net size, number of epochs, and patience for demonstration purposes
>
> - Net sizes `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])
```

11.6.2 Modify hyperparameter of type factor

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

11.6.3 Optimizers

Optimizers are described in Section 14.6.1.

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

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

11.7.1 Evaluation

The evaluation procedure requires the specification of two elements:

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

These are described in Section 19.7.1.

The key `"loss_function"` specifies the loss function which is used during the optimization, see Section 14.7.5.

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

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

11.7.2 Metric

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

11.8 Step 8: Calling the SPOT Function

11.8.1 Preparing the SPOT Call

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

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

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

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

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

11.8.2 The Objective Function `fun_torch`

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

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

11.8.3 Starting the Hyperparameter Tuning

```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,
```

```python
                         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, 'p
Epoch: 1 |


MulticlassAccuracy: 0.1001666635274887 | Loss: 2.3063041612307229 | Acc: 0.1001666666666667.
Epoch: 2 |


MulticlassAccuracy: 0.1001666635274887 | Loss: 2.2886285804112751 | Acc: 0.1001666666666667.
Epoch: 3 |


MulticlassAccuracy: 0.1001666635274887 | Loss: 2.2764437289237978 | Acc: 0.1001666666666667.
Epoch: 4 |


MulticlassAccuracy: 0.1001666635274887 | Loss: 2.2651307185490928 | Acc: 0.1001666666666667.
Epoch: 5 |


MulticlassAccuracy: 0.1001666635274887 | Loss: 2.2538433388074237 | Acc: 0.1001666666666667.
Epoch: 6 |


MulticlassAccuracy: 0.1001666635274887 | Loss: 2.2420443340937295 | Acc: 0.1001666666666667.
Epoch: 7 |


MulticlassAccuracy: 0.1001666635274887 | Loss: 2.2283663109143577 | Acc: 0.1001666666666667.
Epoch: 8 |
```

```
MulticlassAccuracy: 0.1001249998807907 | Loss: 2.2136417441368104 | Acc: 0.1001250000000000.
Returned to Spot: Validation loss: 2.2136417441368104

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

MulticlassAccuracy: 0.0992916673421860 | Loss: 2.3118297967116037 | Acc: 0.0992916666666667.
Epoch: 2 |

MulticlassAccuracy: 0.0992083325982094 | Loss: 2.3051748133500416 | Acc: 0.0992083333333333.
Epoch: 3 |

MulticlassAccuracy: 0.0968749970197678 | Loss: 2.2954170982837678 | Acc: 0.0968750000000000.
Epoch: 4 |

MulticlassAccuracy: 0.0960833355784416 | Loss: 2.2838851950963339 | Acc: 0.0960833333333333.
Returned to Spot: Validation loss: 2.283885195096334

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

MulticlassAccuracy: 0.2995416522026062 | Loss: 2.0943924070497353 | Acc: 0.2995416666666667.
Epoch: 2 |

MulticlassAccuracy: 0.3963333368301392 | Loss: 1.8536794082919756 | Acc: 0.3963333333333333.
Epoch: 3 |

MulticlassAccuracy: 0.5218333601951599 | Loss: 1.6231919860740502 | Acc: 0.5218333333333334.
Epoch: 4 |

MulticlassAccuracy: 0.5931666493415833 | Loss: 1.4076887432460983 | Acc: 0.5931666666666666.
Epoch: 5 |

MulticlassAccuracy: 0.6184999942779541 | Loss: 1.2448123300584655 | Acc: 0.6185000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.6383749842643738 | Loss: 1.1295941969944785 | Acc: 0.6383750000000000.
Epoch: 7 |
```

```
MulticlassAccuracy: 0.6585833430290222 | Loss: 1.0449522113585845 | Acc: 0.6585833333333333.
Epoch: 8 |

MulticlassAccuracy: 0.6753333210945129 | Loss: 0.9797260941676795 | Acc: 0.6753333333333333.
Returned to Spot: Validation loss: 0.9797260941676795

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

MulticlassAccuracy: 0.0987500026822090 | Loss: 2.3302547915180525 | Acc: 0.0987500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.0987500026822090 | Loss: 2.3138896652062733 | Acc: 0.0987500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.0987500026822090 | Loss: 2.3003770700891812 | Acc: 0.0987500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.0987500026822090 | Loss: 2.2892366499304773 | Acc: 0.0987500000000000.
Returned to Spot: Validation loss: 2.2892366499304773

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

MulticlassAccuracy: 0.2571666538715363 | Loss: 2.2736563450495404 | Acc: 0.2571666666666667.
Epoch: 2 |

MulticlassAccuracy: 0.3078333437442780 | Loss: 2.2340031821727750 | Acc: 0.3078333333333333.
Epoch: 3 |

MulticlassAccuracy: 0.3332916796207428 | Loss: 2.1902579158147177 | Acc: 0.3332916666666667.
Epoch: 4 |

MulticlassAccuracy: 0.3457500040531158 | Loss: 2.1435050448179247 | Acc: 0.3457500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.3601250052452087 | Loss: 2.0931943265597024 | Acc: 0.3601250000000000.
Epoch: 6 |
```

```
MulticlassAccuracy: 0.3737083375453949 | Loss: 2.0402453082005181 | Acc: 0.3737083333333333.
Epoch: 7 |

MulticlassAccuracy: 0.3874166607856750 | Loss: 1.9851236569086710 | Acc: 0.3874166666666667.
Epoch: 8 |

MulticlassAccuracy: 0.4005833268165588 | Loss: 1.9290288056135179 | Acc: 0.4005833333333333.
Returned to Spot: Validation loss: 1.9290288056135179


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

MulticlassAccuracy: 0.1636666655540466 | Loss: 2.2711122368176779 | Acc: 0.1636666666666667.
Epoch: 2 |

MulticlassAccuracy: 0.1902083307504654 | Loss: 2.2241929117043813 | Acc: 0.1902083333333333.
Epoch: 3 |

MulticlassAccuracy: 0.2845000028610229 | Loss: 2.1802178237835568 | Acc: 0.2845000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.3045833408832550 | Loss: 2.1349357206424076 | Acc: 0.3045833333333333.
Epoch: 5 |

MulticlassAccuracy: 0.2985833287239075 | Loss: 2.0895770474274955 | Acc: 0.2985833333333333.
Epoch: 6 |

MulticlassAccuracy: 0.2969166636466980 | Loss: 2.0438305748701096 | Acc: 0.2969166666666667.
Epoch: 7 |

MulticlassAccuracy: 0.2991249859333038 | Loss: 1.9974628186623256 | Acc: 0.2991250000000000.
Epoch: 8 |

MulticlassAccuracy: 0.3015416562557220 | Loss: 1.9508257422844568 | Acc: 0.3015416666666667.
Returned to Spot: Validation loss: 1.9508257422844568
spotPython tuning: 0.9797260941676795 [#####-----] 50.47%
```

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

MulticlassAccuracy: 0.0997499972581863 | Loss: 2.2544804678658643 | Acc: 0.0997500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.1156666651368141 | Loss: 2.1755044022500516 | Acc: 0.1156666666666667.
Epoch: 3 |

MulticlassAccuracy: 0.1630833297967911 | Loss: 2.0967100175817808 | Acc: 0.1630833333333333.
Epoch: 4 |

MulticlassAccuracy: 0.2820833325386047 | Loss: 2.0045724806686240 | Acc: 0.2820833333333334.
Returned to Spot: Validation loss: 2.004572480668624

spotPython tuning: 0.9797260941676795 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x2b107bb50>
```

11.9 Step 9: Tensorboard

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

11.10 Step 10: Results

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

```
SAVE = False
LOAD = False

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

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

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

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

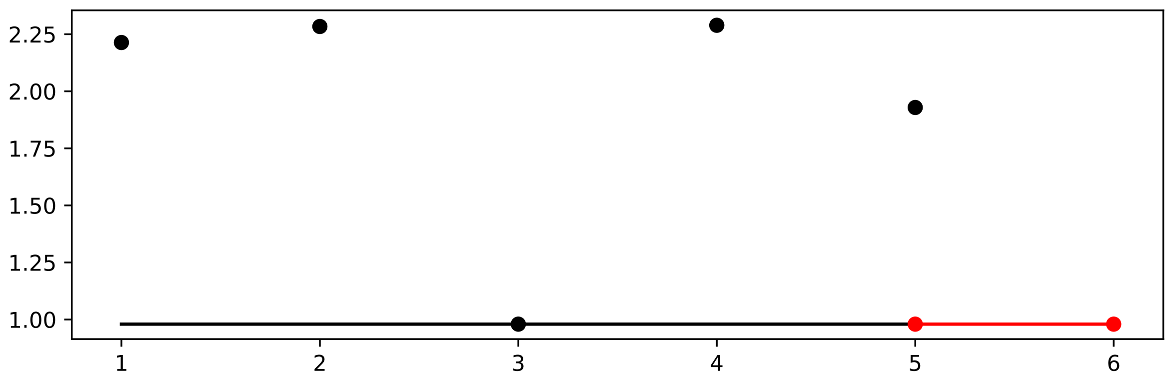


Figure 11.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- Print the results

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

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

11.10.1 Show variable importance

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

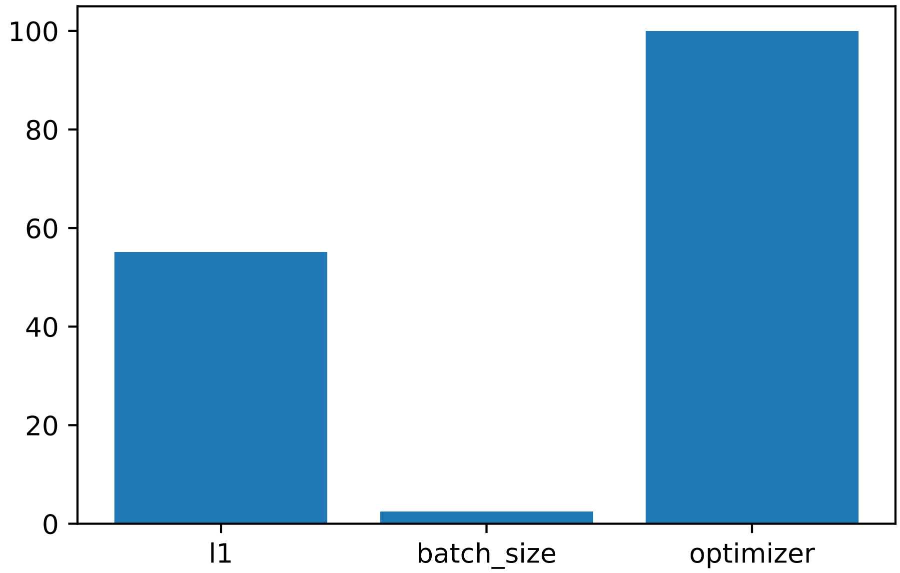


Figure 11.2: Variable importance plot, threshold 0.025.

11.10.2 Get the Tuned Architecture (SPOT Results)

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

```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
```

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

```
  )
)
```

11.10.3 Get Default Hyperparameters

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

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

11.10.4 Evaluation of the Default and the Tuned Architectures

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

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

```
Epoch: 1 |
```

```
MulticlassAccuracy: 0.1989583373069763 | Loss: 2.0225593659083048 | Acc: 0.1989583333333333.
Epoch: 2 |

MulticlassAccuracy: 0.5074999928474426 | Loss: 1.5083103354771932 | Acc: 0.5075000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.5678333044052124 | Loss: 1.2364873363971711 | Acc: 0.5678333333333333.
Epoch: 4 |

MulticlassAccuracy: 0.5996666550636292 | Loss: 1.1032714655796687 | Acc: 0.5996666666666667.
Epoch: 5 |

MulticlassAccuracy: 0.6286249756813049 | Loss: 1.0228628550370533 | Acc: 0.6286250000000000.
Epoch: 6 |

MulticlassAccuracy: 0.6478333473205566 | Loss: 0.9639184764623642 | Acc: 0.6478333333333334.
Epoch: 7 |

MulticlassAccuracy: 0.6657500267028809 | Loss: 0.9191961910923322 | Acc: 0.6657500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.6818333268165588 | Loss: 0.8826866638064385 | Acc: 0.6818333333333333.
Returned to Spot: Validation loss: 0.8826866638064385
```

```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: 0.6639999747276306 | Loss: 0.8956424621105195 | Acc: 0.6640000000000000.
Final evaluation: Validation loss: 0.8956424621105195
Final evaluation: Validation metric: 0.6639999747276306
----------------------------------------------

(0.8956424621105195, nan, tensor(0.6640))
```

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): 2.270

MulticlassAccuracy: 0.1362500041723251 | Loss: 2.1294869047602019 | Acc: 0.1362500000000000.
Epoch: 2 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.069

MulticlassAccuracy: 0.3404583334922791 | Loss: 1.9031155826052031 | Acc: 0.3404583333333333.
Epoch: 3 |

Batch: 10000. Batch Size: 2. Training Loss (running): 1.831

MulticlassAccuracy: 0.3936666548252106 | Loss: 1.6664898740202188 | Acc: 0.3936666666666667.
Epoch: 4 |

Batch: 10000. Batch Size: 2. Training Loss (running): 1.602

MulticlassAccuracy: 0.4684999883174896 | Loss: 1.4569098324825367 | Acc: 0.4685000000000000.
Epoch: 5 |

Batch: 10000. Batch Size: 2. Training Loss (running): 1.403

MulticlassAccuracy: 0.5705416798591614 | Loss: 1.2849842578818400 | Acc: 0.5705416666666666.
Epoch: 6 |

Batch: 10000. Batch Size: 2. Training Loss (running): 1.247
```

```
MulticlassAccuracy: 0.6261249780654907 | Loss: 1.1523726394530385 | Acc: 0.6261250000000000.
Epoch: 7 |

Batch: 10000. Batch Size: 2. Training Loss (running): 1.118

MulticlassAccuracy: 0.6594166755676270 | Loss: 1.0527777444332218 | Acc: 0.6594166666666667.
Epoch: 8 |

Batch: 10000. Batch Size: 2. Training Loss (running): 1.035

MulticlassAccuracy: 0.6727499961853027 | Loss: 0.9787188845334264 | Acc: 0.6727500000000000.
Returned to Spot: Validation loss: 0.9787188845334264
```

```
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: 0.6697000265121460 | Loss: 0.9877570300258696 | Acc: 0.6697000000000000.
Final evaluation: Validation loss: 0.9877570300258696
Final evaluation: Validation metric: 0.669700026512146
----------------------------------------------

(0.9877570300258696, nan, tensor(0.6697))
```

11.10.5 Detailed Hyperparameter Plots

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

```
l1:  55.16166620423628
batch_size:  2.498961516713552
optimizer:  100.0
```

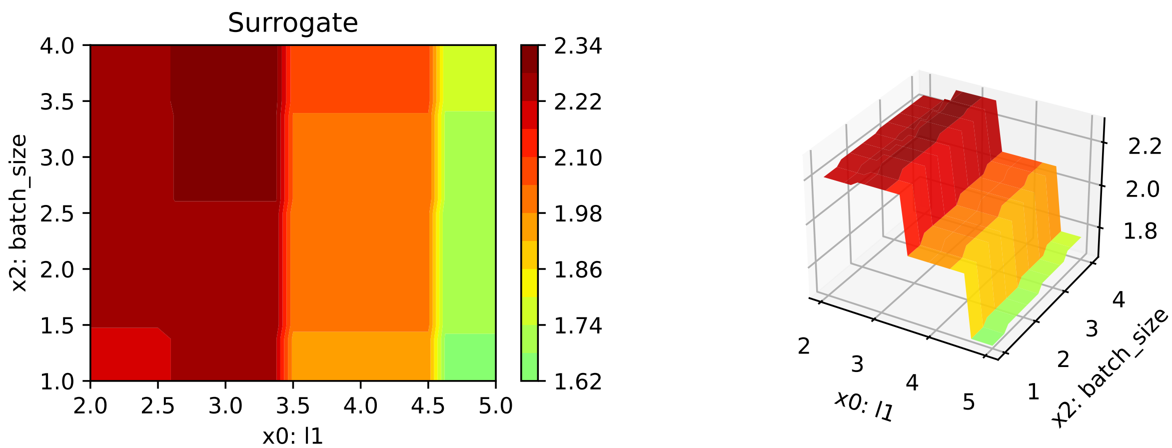
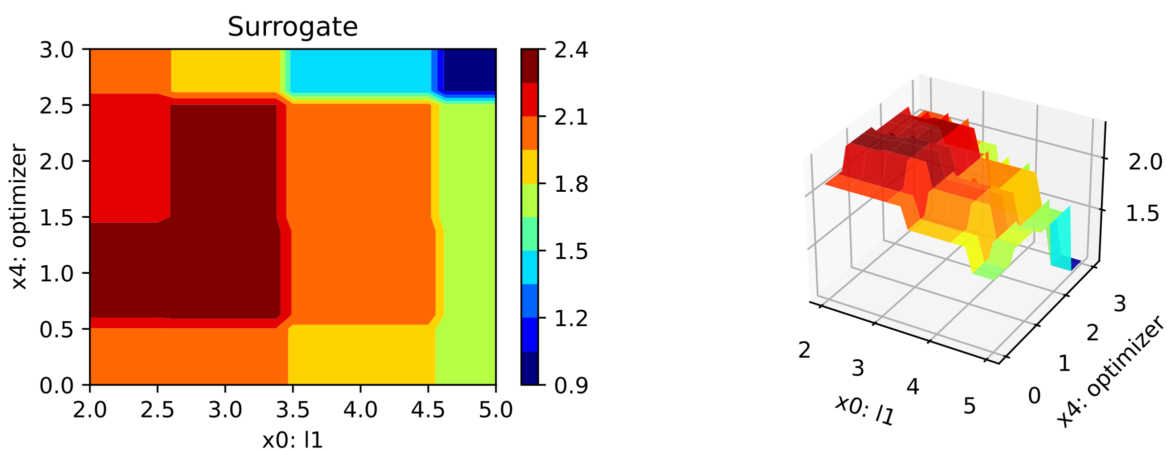
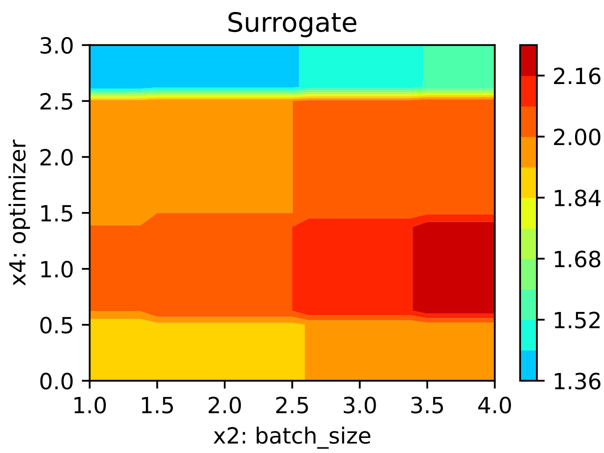
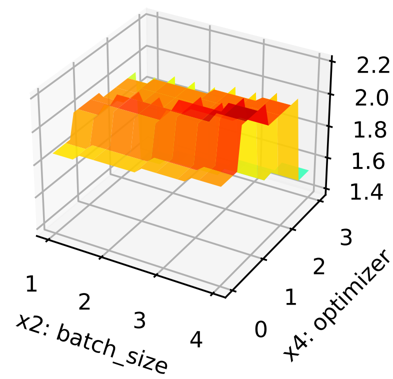


Figure 11.3: Contour plots.







11.10.6 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

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

Parallel coordinates plots

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

11.10.7 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

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

12 HPT: PyTorch With cifar10 Data

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

This document refers to the following software versions:

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

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

```
spotPython                 0.2.46
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
```

12.1 Step 1: Setup

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

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

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

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_2023-06-27_02-49-56
```

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

`spotPython` uses a Python dictionary for storing the information required for the hyperparameter tuning process, which was described in Section 14.2.

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

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

12.3 Step 3: PyTorch Data Loading

12.3.1 Load Data Cifar10 Data

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

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

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

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

```
Files already downloaded and verified
```

```
Files already downloaded and verified
```

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

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

12.4 Step 4: Specification of the Preprocessing Model

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

12.5 Step 5: Select Model (algorithm) and core_model_hyper_dict

12.5.1 Implementing a Configurable Neural Network With spotPython

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

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

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

12.5.2 The Search Space

12.5.3 Configuring the Search Space With spotPython

12.5.3.1 The `hyper_dict` Hyperparameters for the Selected Algorithm

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

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

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

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

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

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

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

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

12.6.1.1 Modify Hyperparameters of Type numeric and integer (boolean)

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

> Caution: Small net size, number of epochs, and patience for demonstration purposes
>
> - Net sizes `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])
```

12.6.2 Modify hyperparameter of type factor

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

12.6.3 Optimizers

Optimizers can be selected as described in Section 19.6.2.

Optimizers are described in Section 14.6.1.

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

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

12.7.1 Evaluation

The evaluation procedure requires the specification of two elements:

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

These are described in Section 19.7.1.

The key `"loss_function"` specifies the loss function which is used during the optimization, see Section 14.7.5.

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

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

12.7.2 Metric

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

12.8 Step 8: Calling the SPOT Function

12.8.1 Preparing the SPOT Call

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

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

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

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

12.8.2 The Objective Function `fun_torch`

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

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

12.8.3 Starting the Hyperparameter Tuning

```
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, 'p
Epoch: 1 |


MulticlassAccuracy: 0.0981499999761581 | Loss: 2.3175262256622315 | Acc: 0.0981500000000000.
Epoch: 2 |
```

```
MulticlassAccuracy: 0.0981499999761581 | Loss: 2.3162806632995605 | Acc: 0.0981500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.0981499999761581 | Loss: 2.3148137495040895 | Acc: 0.0981500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.0981499999761581 | Loss: 2.3130374753952028 | Acc: 0.0981500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.0981499999761581 | Loss: 2.3108868780136107 | Acc: 0.0981500000000000.
Epoch: 6 |

MulticlassAccuracy: 0.0981499999761581 | Loss: 2.3082428007125855 | Acc: 0.0981500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.0990500003099442 | Loss: 2.3052609569549563 | Acc: 0.0990500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.1070000007748604 | Loss: 2.3022833627700807 | Acc: 0.1070000000000000.
Returned to Spot: Validation loss: 2.3022833627700807

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

MulticlassAccuracy: 0.0999500006437302 | Loss: 2.3301803409576416 | Acc: 0.0999500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.0999500006437302 | Loss: 2.3280342610836029 | Acc: 0.0999500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.0999500006437302 | Loss: 2.3258641382217409 | Acc: 0.0999500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.0999500006437302 | Loss: 2.3235516609191893 | Acc: 0.0999500000000000.
Returned to Spot: Validation loss: 2.3235516609191893

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

```
MulticlassAccuracy: 0.1203500032424927 | Loss: 2.2994544739007949 | Acc: 0.1203500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.1597000062465668 | Loss: 2.2763449429035187 | Acc: 0.1597000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.1854999959468842 | Loss: 2.2348965238928793 | Acc: 0.1855000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.1997999995946884 | Loss: 2.1900893228054046 | Acc: 0.1998000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.2226500064134598 | Loss: 2.1496455568909645 | Acc: 0.2226500000000000.
Epoch: 6 |

MulticlassAccuracy: 0.2330500036478043 | Loss: 2.1185323487639427 | Acc: 0.2330500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.2430499941110611 | Loss: 2.0929177416026592 | Acc: 0.2430500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.2510499954223633 | Loss: 2.0706511720478535 | Acc: 0.2510500000000000.
Returned to Spot: Validation loss: 2.0706511720478535

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

MulticlassAccuracy: 0.0989999994635582 | Loss: 2.3310777954816819 | Acc: 0.0990000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.0989999994635582 | Loss: 2.3290804311275481 | Acc: 0.0990000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.0989999994635582 | Loss: 2.3268079236268999 | Acc: 0.0990000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.0989999994635582 | Loss: 2.3240199430942536 | Acc: 0.0990000000000000.
Returned to Spot: Validation loss: 2.3240199430942536

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

```
MulticlassAccuracy: 0.1013500019907951 | Loss: 2.3055892971038818 | Acc: 0.1013500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.1013500019907951 | Loss: 2.3049407145500185 | Acc: 0.1013500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.1013500019907951 | Loss: 2.3041414135932921 | Acc: 0.1013500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.1039000004529953 | Loss: 2.3029593177795409 | Acc: 0.1039000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.1214499995112419 | Loss: 2.3010701871871948 | Acc: 0.1214500000000000.
Epoch: 6 |

MulticlassAccuracy: 0.1400499939918518 | Loss: 2.2984546786308289 | Acc: 0.1400500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.1495999991893768 | Loss: 2.2952054484367372 | Acc: 0.1496000000000000.
Epoch: 8 |

MulticlassAccuracy: 0.1537500023841858 | Loss: 2.2905507787704469 | Acc: 0.1537500000000000.
Returned to Spot: Validation loss: 2.290550778770447


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

MulticlassAccuracy: 0.1009000018239021 | Loss: 2.3102958086013792 | Acc: 0.1009000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.1009000018239021 | Loss: 2.3083570036888124 | Acc: 0.1009000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.1009000018239021 | Loss: 2.3062401810646058 | Acc: 0.1009000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.1028499975800514 | Loss: 2.3033352947235106 | Acc: 0.1028500000000000.
Epoch: 5 |
```

```
MulticlassAccuracy: 0.1263500005006790 | Loss: 2.2985390166282653 | Acc: 0.1263500000000000.
Epoch: 6 |

MulticlassAccuracy: 0.1367000043392181 | Loss: 2.2923821520805361 | Acc: 0.1367000000000000.
Epoch: 7 |

MulticlassAccuracy: 0.1333000063896179 | Loss: 2.2849505395889280 | Acc: 0.1333000000000000.
Epoch: 8 |

MulticlassAccuracy: 0.1312499940395355 | Loss: 2.2765993299484255 | Acc: 0.1312500000000000.
Returned to Spot: Validation loss: 2.2765993299484255
spotPython tuning: 2.0706511720478535 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x2a28c3af0>
```

12.9 Step 9: Tensorboard

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

12.10 Step 10: Results

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

```
SAVE = False
LOAD = False

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

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

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

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

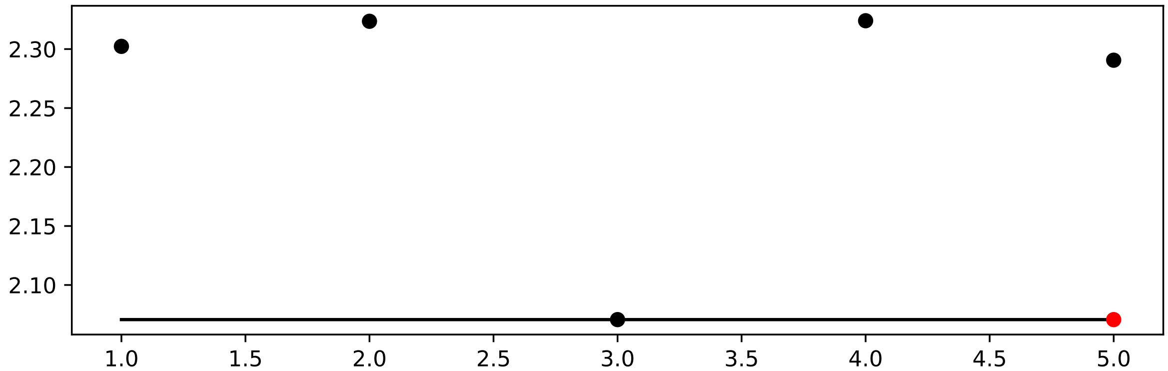


Figure 12.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- Print the results

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

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

12.10.1 Show variable importance

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

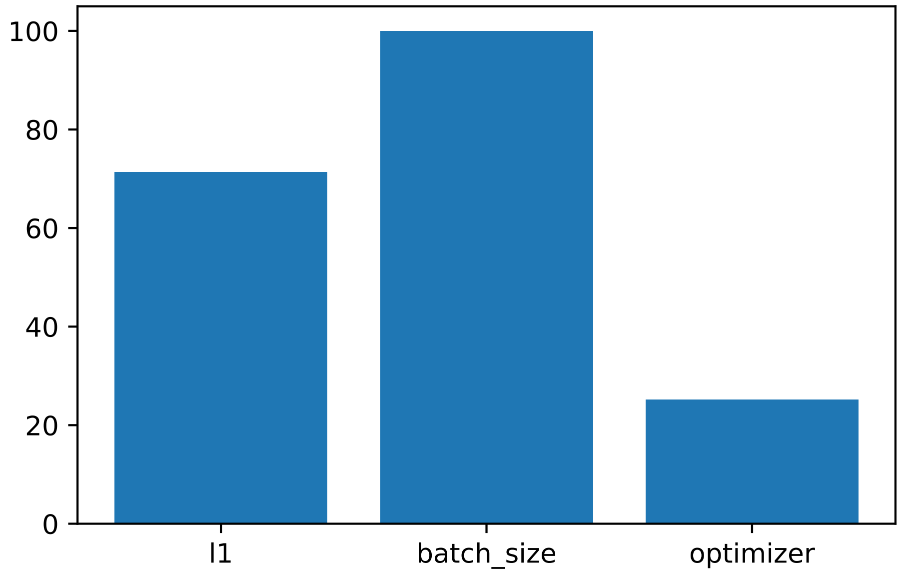


Figure 12.2: Variable importance plot, threshold 0.025.

12.10.2 Get the Tuned Architecture (SPOT Results)

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

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

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

12.10.3 Evaluation of the Tuned Architecture

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

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

```
Epoch: 1 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.307

MulticlassAccuracy: 0.1280999928712845 | Loss: 2.2973201494932174 | Acc: 0.1281000000000000.
Epoch: 2 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.288

MulticlassAccuracy: 0.1253499984741211 | Loss: 2.2677038742899893 | Acc: 0.1253500000000000.
Epoch: 3 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.255

MulticlassAccuracy: 0.1430500000715256 | Loss: 2.2291841602206230 | Acc: 0.1430500000000000.
Epoch: 4 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.216

MulticlassAccuracy: 0.1508499979972839 | Loss: 2.1870988730549814 | Acc: 0.1508500000000000.
Epoch: 5 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.175
```

```
MulticlassAccuracy: 0.1669500023126602 | Loss: 2.1472811633825302 | Acc: 0.1669500000000000.
Epoch: 6 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.136

MulticlassAccuracy: 0.1912499964237213 | Loss: 2.1120697822451593 | Acc: 0.1912500000000000.
Epoch: 7 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.104

MulticlassAccuracy: 0.2257499992847443 | Loss: 2.0821929213941099 | Acc: 0.2257500000000000.
Epoch: 8 |

Batch: 10000. Batch Size: 2. Training Loss (running): 2.074

MulticlassAccuracy: 0.2506000101566315 | Loss: 2.0564927400290967 | Acc: 0.2506000000000000.
Returned to Spot: Validation loss: 2.0564927400290967
```

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: 0.2572999894618988 | Loss: 2.0522853253602982 | Acc: 0.2573000000000000.
Final evaluation: Validation loss: 2.052285325360298
Final evaluation: Validation metric: 0.2572999894618988
----------------------------------------------
```

```
(2.052285325360298, nan, tensor(0.2573))
```

12.10.4 Cross-validated Evaluations

Caution: Cross-validated Evaluations

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

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

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

12.10.5 Detailed Hyperparameter Plots

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

```
l1:  71.4182466173975
batch_size:  100.0
optimizer:  25.164335115264716
```

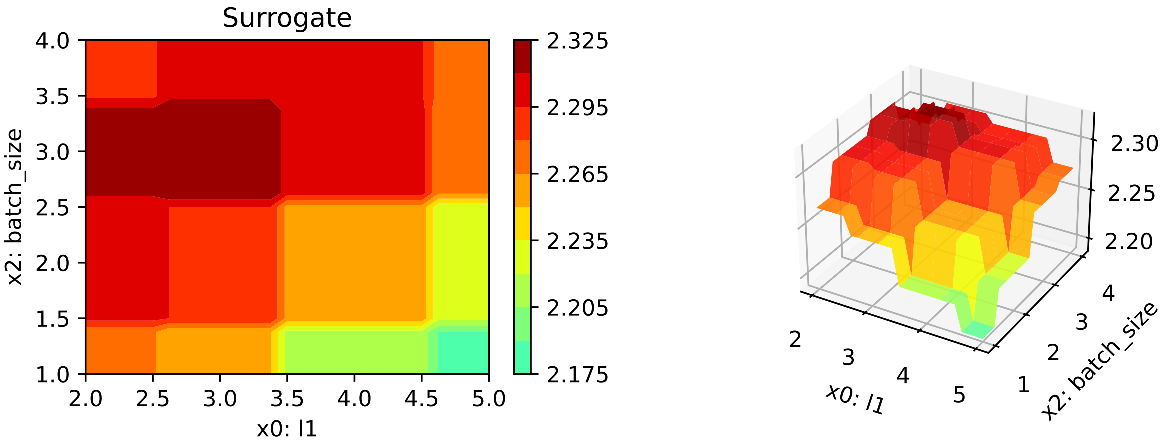
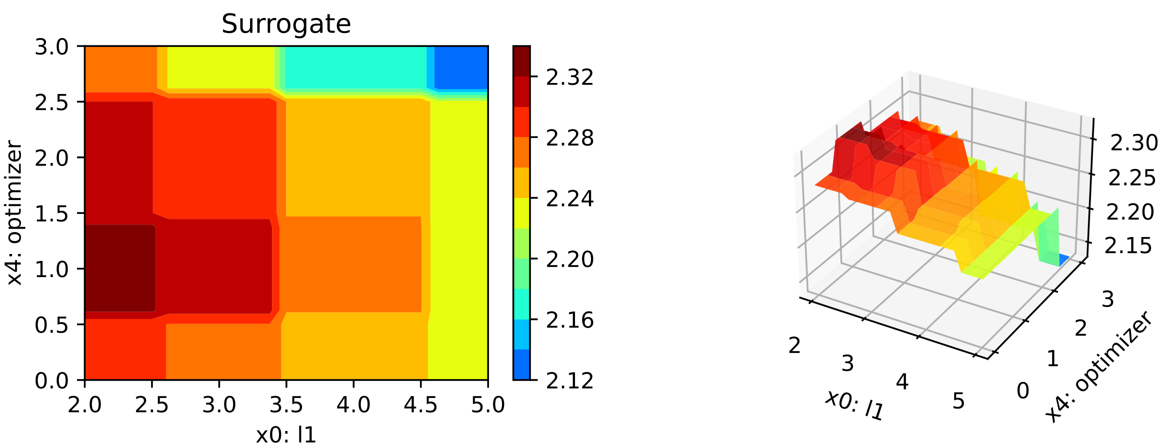
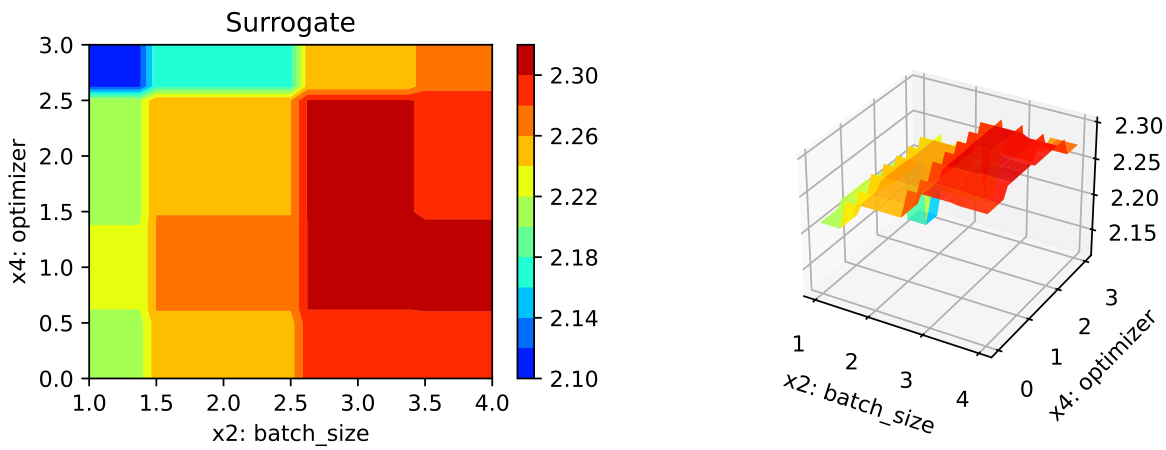


Figure 12.3: Contour plots.





12.10.6 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

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

Parallel coordinates plots

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

12.10.7 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

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

13 HPT: River

River is a Python library for online machine learning (Montiel et al. 2021). It aims to be the most user-friendly library for doing machine learning on streaming data. River is the result of a merger between creme and scikit-multiflow.

13.1 Step 1: Setup

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

> Caution: Run time and initial design size should be increased for real experiments
>
> - MAX_TIME is set to one minute for demonstration purposes. For real experiments, this should be increased to at least 1 hour.
> - INIT_SIZE is set to 5 for demonstration purposes. For real experiments, this should be increased to at least 10.
> - K is set to 0.1 for demonstration purposes. For real experiments, this should be increased to at least 1.

```
MAX_TIME = 1
INIT_SIZE = 5
K = .1
```

```
10-river_bartz09_1min_5init_2023-06-27_03-03-42
```

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

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

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

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

```
spotPython                 0.2.46
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
```

13.2 Step 2: Initialization of the `fun_control` Dictionary

```
from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="regression",
 tensorboard_path=None)
```

13.3 Step 3: Load the Friedman Drift Data

```
horizon = 7*24
k = K
n_total = int(k*100_000)
n_samples = n_total
p_1 = int(k*25_000)
p_2 = int(k*50_000)
position=(p_1, p_2)
n_train = 1_000
a = n_train + p_1 - 12
b = a + 12
```

- Since we also need a `river` version of the data below for plotting the model, the corresponding data set is generated here. Note: `spotRiver` uses the `train` and `test` data sets, while `river` uses the `X` and `y` data sets

```
from river.datasets import synth
import pandas as pd
dataset = synth.FriedmanDrift(
   drift_type='gra',
   position=position,
     seed=123
)
data_dict = {key: [] for key in list(dataset.take(1))[0][0].keys()}
data_dict["y"] = []
for x, y in dataset.take(n_total):
    for key, value in x.items():
        data_dict[key].append(value)
    data_dict["y"].append(y)
df = pd.DataFrame(data_dict)
# Add column names x1 until x10 to the first 10 columns of the dataframe and the column 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})
```

13.4 Step 4: Specification of the Preprocessing Model

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

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

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

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

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

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

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

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

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

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

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

13.6.1 Modify hyperparameter of type factor

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

13.6.2 Modify hyperparameter of type numeric and integer (boolean)

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

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

There are three metrics:

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

```
import numpy as np
from river import metrics
from sklearn.metrics import mean_absolute_error


from spotRiver.fun.hyperriver import HyperRiver
fun = HyperRiver(seed=123, log_level=50).fun_oml_horizon
weights = np.array([1, 1/1000, 1/1000])*10_000.0
horizon = 7*24
```

```python
oml_grace_period = 2
step = 100
weight_coeff = 1.0

fun_control.update({
               "horizon": horizon,
               "oml_grace_period": oml_grace_period,
               "weights": weights,
               "step": step,
               "log_level": 50,
               "weight_coeff": weight_coeff,
               "metric_river": metrics.MAE(),
               "metric_sklearn": mean_absolute_error
               })
```

13.8 Step 8: Calling the SPOT Function

13.8.1 Prepare the SPOT Parameters

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

```python
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")
```

```python
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
```

13.8.2 Run the Spot Optimizer

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

```python
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)
```

```python
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: 2.180527079238113 [###-------] 31.15%

spotPython tuning: 2.180527079238113 [######----] 58.14%

spotPython tuning: 2.180527079238113 [########--] 75.36%

spotPython tuning: 2.180527079238113 [##########] 95.66%

spotPython tuning: 2.180527079238113 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x1034ce5c0>
```

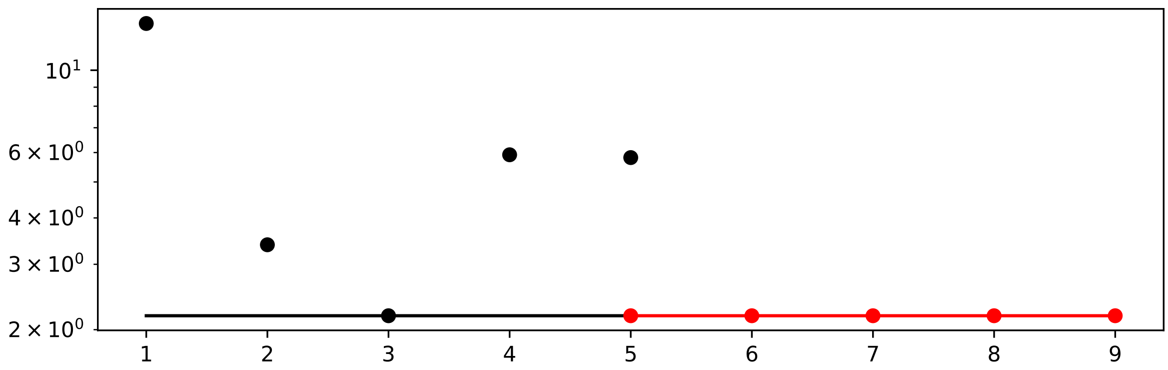
13.9 Step 9: Results

```
import pickle
SAVE = False
LOAD = False
```

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

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

- Show the Progress of the hyperparameter tuning:

```
spot_tuner.plot_progress(log_y=True, filename="./figures/" + experiment_name+"_progress.pd
```



- Print the Results

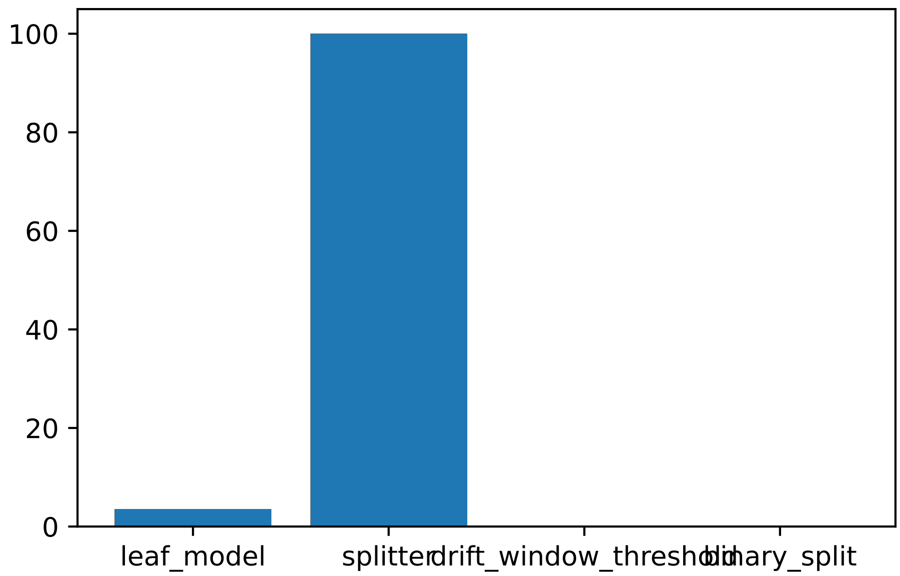
```
print(gen_design_table(fun_control=fun_control, spot=spot_tuner))
```

```
| name                   | type   | default          |     lower |      upper |
|------------------------|--------|------------------|-----------|------------|----------------
| grace_period           | int    | 200              |      10.0 |     1000.0 |
| max_depth              | int    | 20               |       2.0 |       20.0 |
| delta                  | float  | 1e-07            |     1e-10 |      1e-06 | 4.068723023437
| tau                    | float  | 0.05             |      0.01 |        0.1 |    0.048426009
| leaf_prediction        | factor | mean             |       0.0 |        2.0 |
| leaf_model             | factor | LinearRegression |       0.0 |        2.0 |
| model_selector_decay   | float  | 0.95             |       0.9 |       0.99 |     0.97071323
| splitter               | factor | EBSTSplitter     |       0.0 |        2.0 |
| min_samples_split      | int    | 5                |       2.0 |       10.0 |
```

```
| bootstrap_sampling      | factor | 0                    |      0.0 |       1.0 |
| drift_window_threshold | int    | 300                  |    100.0 |     500.0 |
| switch_significance     | float  | 0.05                 |     0.01 |       0.1 |     0.040370639
| binary_split            | factor | 0                    |      0.0 |       1.0 |
| max_size                | float  | 500.0                |    100.0 |    1000.0 |      454.140654
| memory_estimate_period | int    | 1000000              | 100000.0 | 1000000.0 |               9
| stop_mem_management     | factor | 0                    |      0.0 |       1.0 |
| remove_poor_attrs       | factor | 0                    |      0.0 |       1.0 |
| merit_preprune          | factor | 0                    |      0.0 |       0.0 |
```

13.9.1 Show variable importance

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



13.9.2 Build and Evaluate HTR Model with Tuned Hyperparameters

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

13.9.3 The Large Data Set (k=0.2)

Caution: Increased Friedman-Drift Data Set

- The Friedman-Drift Data Set is increased by a factor of two to show the transferability of the hyperparameter tuning results.
- Larger values of `k` lead to a longer run time.

```
horizon = 7*24
k = .2
n_total = int(k*100_000)
n_samples = n_total
p_1 = int(k*25_000)
p_2 = int(k*50_000)
position=(p_1, p_2)
n_train = 1_000
a = n_train + p_1 - 12
b = a + 12
dataset = synth.FriedmanDrift(
   drift_type='gra',
   position=position,
     seed=123
)
data_dict = {key: [] for key in list(dataset.take(1))[0][0].keys()}
data_dict["y"] = []
for x, y in dataset.take(n_total):
    for key, value in x.items():
        data_dict[key].append(value)
    data_dict["y"].append(y)
df = pd.DataFrame(data_dict)
# Add column names x1 until x10 to the first 10 columns of the dataframe and the column 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})
```

13.9.4 Get Default Hyperparameters

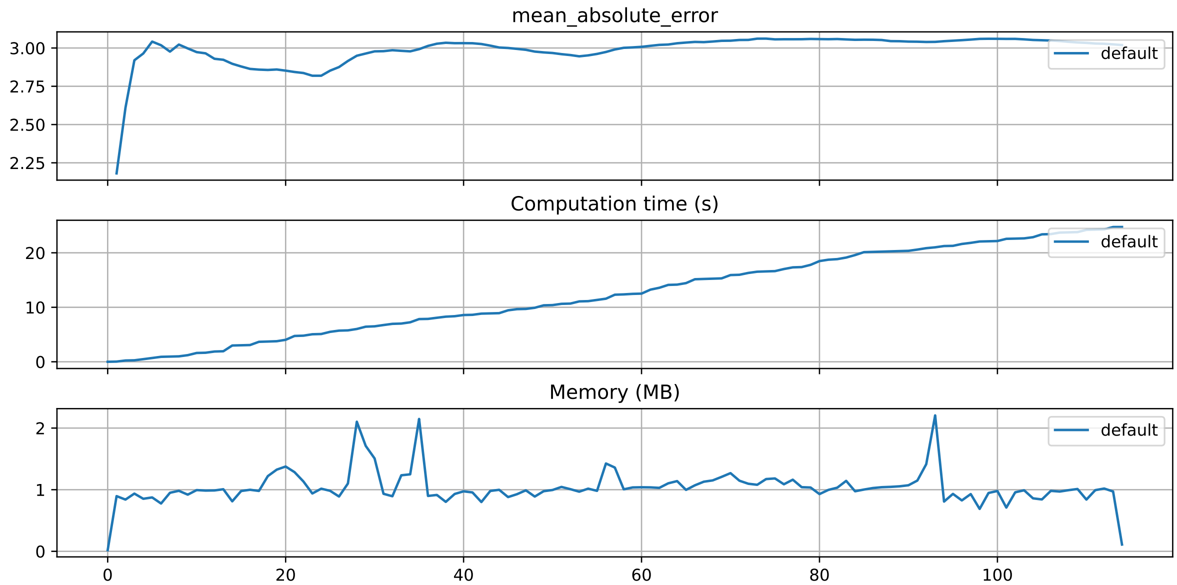
```
# fun_control was modified, we generate a new one with the original
# default hyperparameters
from spotPython.hyperparameters.values import get_one_core_model_from_X
fc = fun_control
fc.update({"core_model_hyper_dict":
    hyper_dict[fun_control["core_model"].__name__]})
model_default = get_one_core_model_from_X(X_start,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

```
HoeffdingAdaptiveTreeRegressor (
  grace_period=200
  max_depth=1048576
  delta=1e-07
  tau=0.05
  leaf_prediction="mean"
  leaf_model=LinearRegression (
    optimizer=SGD (
      lr=Constant (
        learning_rate=0.01
      )
    )
    loss=Squared ()
    l2=0.
    l1=0.
    intercept_init=0.
    intercept_lr=Constant (
      learning_rate=0.01
    )
    clip_gradient=1e+12
    initializer=Zeros ()
  )
  model_selector_decay=0.95
  nominal_attributes=None
  splitter=EBSTSplitter ()
  min_samples_split=5
  bootstrap_sampling=0
```

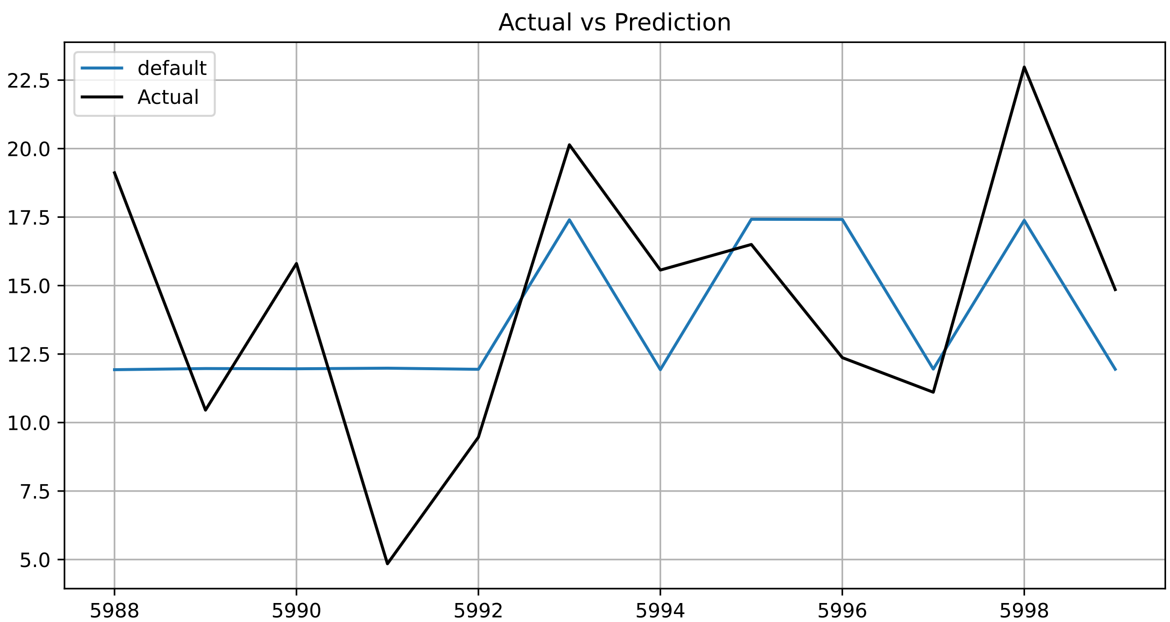
```
  drift_window_threshold=300
  drift_detector=ADWIN (
    delta=0.002
    clock=32
    max_buckets=5
    min_window_length=5
    grace_period=10
  )
  switch_significance=0.05
  binary_split=0
  max_size=500.
  memory_estimate_period=1000000
  stop_mem_management=0
  remove_poor_attrs=0
  merit_preprune=0
  seed=None
)
```

```
from spotRiver.evaluation.eval_bml import eval_oml_horizon

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

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



mean_absolute_error
3.00
2.75
2.50
2.25
default
Computation time (s)
20
10
0
default
Memory (MB)
2
1
0
default
0
20
40
60
80
100



Actual vs Prediction
default
Actual
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
5988
5990
5992
5994
5996
5998

13.9.5 Get SPOT Results

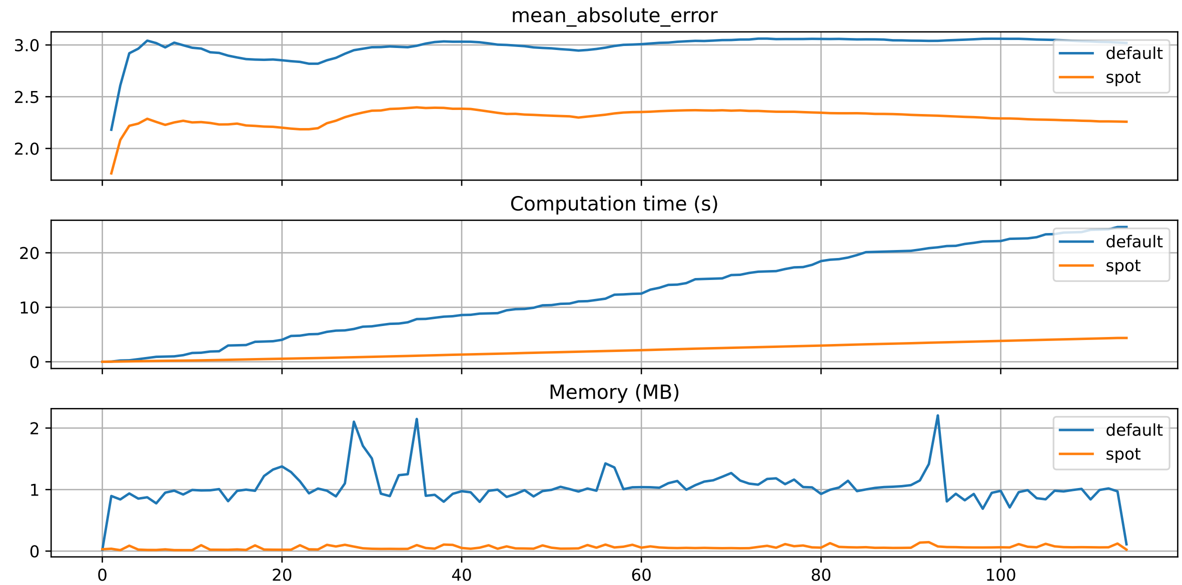
```
from spotPython.hyperparameters.values import get_one_core_model_from_X
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
model_spot = get_one_core_model_from_X(X, fun_control)
model_spot
```

```
HoeffdingAdaptiveTreeRegressor (
  grace_period=657
  max_depth=256
  delta=4e-08
  tau=0.048426
  leaf_prediction="adaptive"
  leaf_model=LinearRegression (
    optimizer=SGD (
      lr=Constant (
        learning_rate=0.01
      )
    )
    loss=Squared ()
    l2=0.
    l1=0.
    intercept_init=0.
    intercept_lr=Constant (
      learning_rate=0.01
    )
    clip_gradient=1e+12
    initializer=Zeros ()
  )
  model_selector_decay=0.970713
  nominal_attributes=None
  splitter=QOSplitter (
    radius=0.25
    allow_multiway_splits=False
  )
  min_samples_split=5
  bootstrap_sampling=1
  drift_window_threshold=166
  drift_detector=ADWIN (
    delta=0.002
    clock=32
    max_buckets=5
```

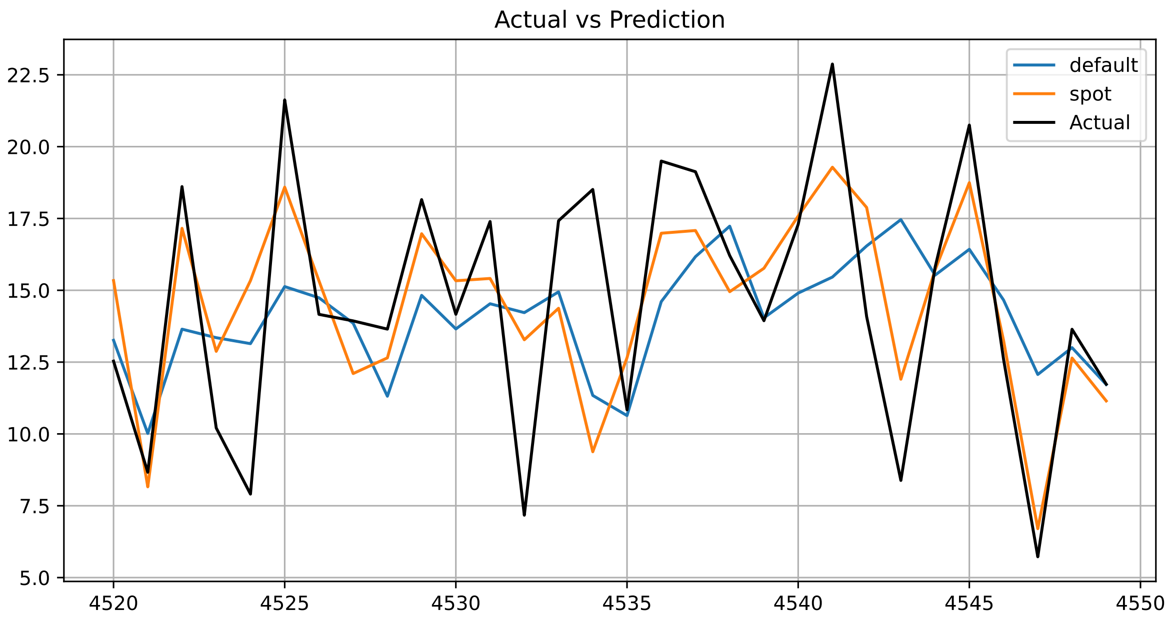
```
    min_window_length=5
    grace_period=10
  )
  switch_significance=0.040371
  binary_split=0
  max_size=454.140654
  memory_estimate_period=910594
  stop_mem_management=1
  remove_poor_attrs=1
  merit_preprune=0
  seed=None
)
```

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

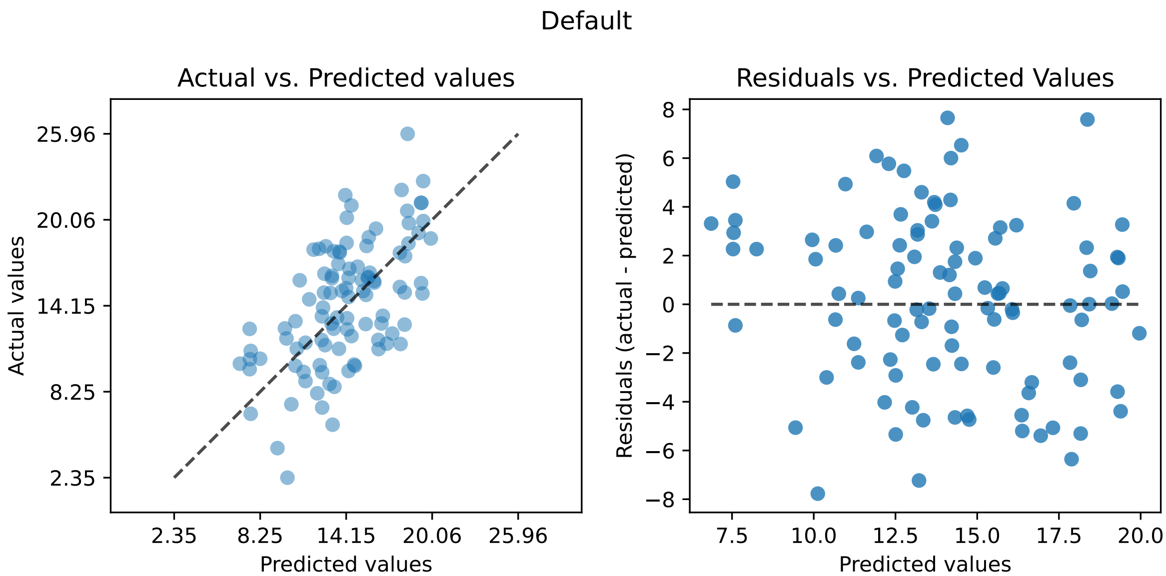
```
df_labels=["default", "spot"]
plot_bml_oml_horizon_metrics(df_eval = [df_eval_default, df_eval_spot], log_y=False, df_la
```

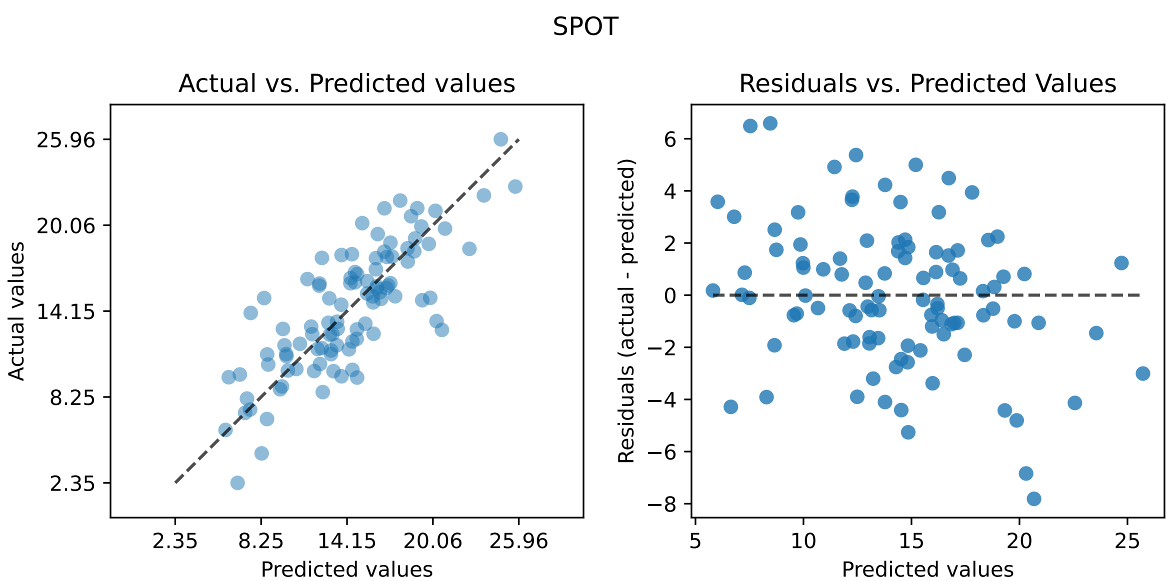



```
a = int(m/2)+20
b = int(m/2)+50
plot_bml_oml_horizon_predictions(df_true = [df_true_default[a:b], df_true_spot[a:b]], targ
```



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





13.9.6 Visualize Regression Trees

```
dataset_f = dataset.take(n_total)
for x, y in dataset_f:
    model_default.learn_one(x, y)
```

> Caution: Large Trees
>
> - Since the trees are large, the visualization is suppressed by default.
> - To visualize the trees, uncomment the following line.

```
# model_default.draw()
```

```
model_default.summary
```

```
{'n_nodes': 35,
 'n_branches': 17,
 'n_leaves': 18,
 'n_active_leaves': 96,
 'n_inactive_leaves': 0,
 'height': 6,
 'total_observed_weight': 39002.0,
 'n_alternate_trees': 21,
 'n_pruned_alternate_trees': 6,
 'n_switch_alternate_trees': 2}
```

13.9.7 Spot Model

```
dataset_f = dataset.take(n_total)
for x, y in dataset_f:
    model_spot.learn_one(x, y)
```

> Caution: Large Trees
>
> - Since the trees are large, the visualization is suppressed by default.
> - To visualize the trees, uncomment the following line.

```
# model_spot.draw()
```

```python
model_spot.summary
```

```
{'n_nodes': 29,
 'n_branches': 14,
 'n_leaves': 15,
 'n_active_leaves': 46,
 'n_inactive_leaves': 0,
 'height': 7,
 'total_observed_weight': 39002.0,
 'n_alternate_trees': 24,
 'n_pruned_alternate_trees': 15,
 '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 |     29 |
| n_branches               |        17 |     14 |
| n_leaves                 |        18 |     15 |
| n_active_leaves          |        96 |     46 |
| n_inactive_leaves        |         0 |      0 |
| height                   |         6 |      7 |
| total_observed_weight    |     39002 |  39002 |
| n_alternate_trees        |        21 |     24 |
| n_pruned_alternate_trees |         6 |     15 |
| n_switch_alternate_trees |         2 |      1 |
```

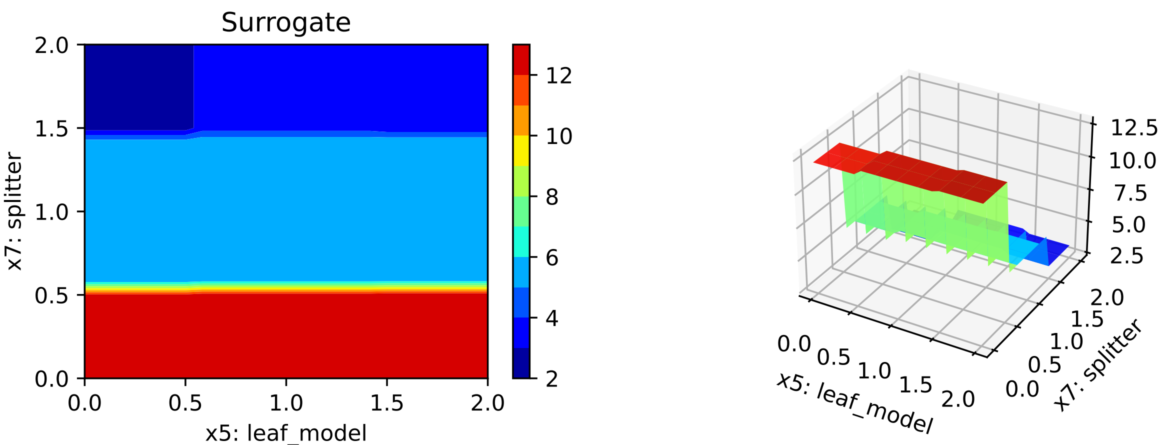
```python
min(spot_tuner.y), max(spot_tuner.y)
```

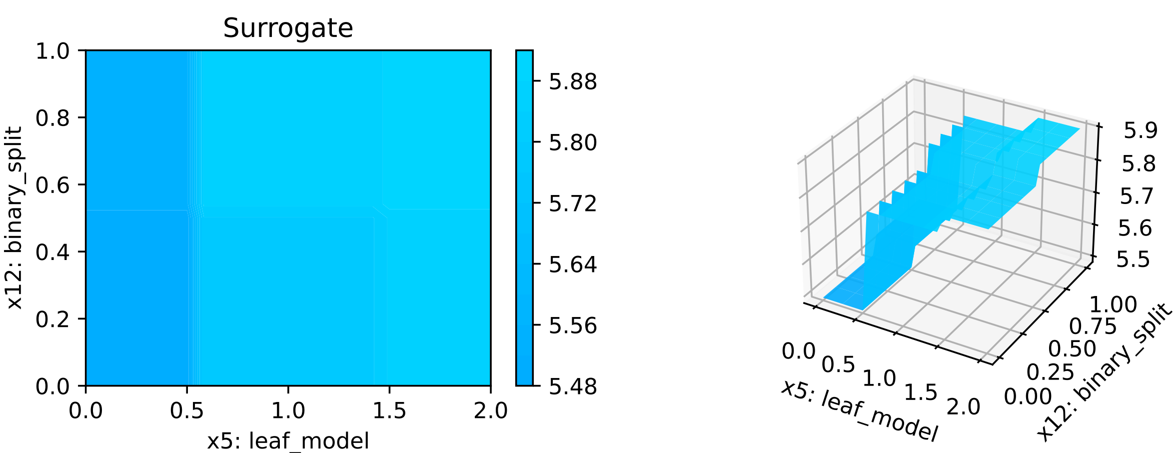
```
(2.180527079238113, 13.363007455405022)
```

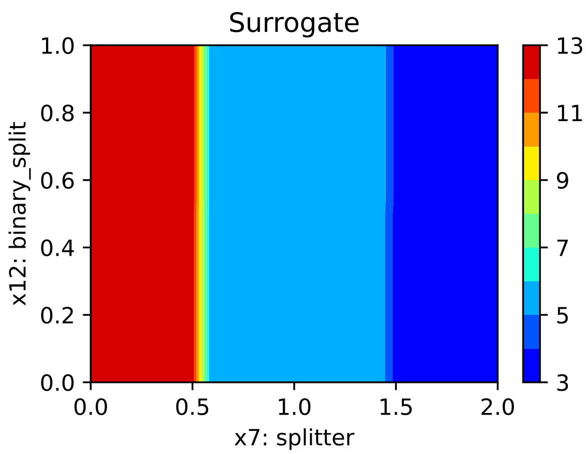
13.9.8 Detailed Hyperparameter Plots

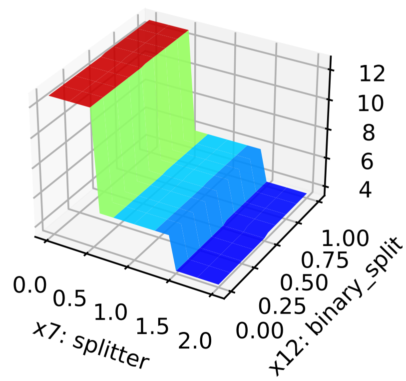
```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

```
leaf_model:  3.534100417251931
splitter:  100.0
binary_split:  0.0728015357077984
```









13.9.9 Parallel Coordinates Plots

```
spot_tuner.parallel_plot()
```

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

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

13.9.10 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

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

14 HPT: PyTorch With spotPython and Ray Tune on CIFAR10

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

This document refers to the following software versions:

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

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

```
spotPython                0.2.46
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].

14.1 Step 1: Setup

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

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

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

Note: Device selection

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

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

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

```
cpu
```

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

[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_2023-06-27_03-27-22
```

14.2 Step 2: Initialization of the `fun_control` Dictionary

`spotPython` uses a Python dictionary for storing the information required for the hyperparameter tuning process. This dictionary is called `fun_control` and is initialized with the function `fun_control_init`. The function `fun_control_init` returns a skeleton dictionary. The dictionary is filled with the required information for the hyperparameter tuning process. It stores the hyperparameter tuning settings, e.g., the deep learning network architecture that should be tuned, the classification (or regression) problem, and the data that is used for the tuning. The dictionary is used as an input for the SPOT function.

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

```
from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="classification",
    tensorboard_path="runs/14_spot_ray_hpt_torch_cifar10",
    device=DEVICE,)
```

14.3 Step 3: PyTorch Data Loading

The data loading process is implemented in the same manner as described in the Section "Data loaders" in PyTorch (2023a). The data loaders are wrapped into the function load_data_cifar10 which is identical to the function load_data in PyTorch (2023a). A global data directory is used, which allows sharing the data directory between different trials. The method load_data_cifar10 is part of the spotPython package and can be imported from spotPython.data.torchdata.

In the following step, the test and train data are added to the dictionary fun_control.

```python
from spotPython.data.torchdata import load_data_cifar10
train, test = load_data_cifar10()
n_samples = len(train)
# add the dataset to the fun_control
fun_control.update({
    "train": train,
    "test": test,
    "n_samples": n_samples})
```

```
Files already downloaded and verified
```

```
Files already downloaded and verified
```

14.4 Step 4: Specification of the Preprocessing Model

After the training and test data are specified and added to the fun_control dictionary, spotPython allows the specification of a data preprocessing pipeline, e.g., for the scaling of the data or for the one-hot encoding of categorical variables. The preprocessing model is called prep_model ("preparation" or pre-processing) and includes steps that are not subject to the hyperparameter tuning process. The preprocessing model is specified in the fun_control dictionary. The preprocessing model can be implemented as a sklearn pipeline. The following code shows a typical preprocessing pipeline:

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

Because the Ray Tune (`ray[tune]`) hyperparameter tuning as described in PyTorch (2023a) does not use a preprocessing model, the preprocessing model is set to `None` here.

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

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

The same neural network model as implemented in the section "Configurable neural network" of the `PyTorch` tutorial (PyTorch 2023a) is used here. We will show the implementation from PyTorch (2023a) in Section 14.5.0.1 first, before the extended implementation with `spotPython` is shown in Section 14.5.0.2.

14.5.0.1 Implementing a Configurable Neural Network With Ray Tune

We used the same hyperparameters that are implemented as configurable in the `PyTorch` tutorial. We specify the layer sizes, namely `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)
```

14.5.0.2 Implementing a Configurable Neural Network With spotPython

`spotPython` implements a class which is similar to the class described in the `PyTorch` tutorial. The class is called `Net_CIFAR10` and is implemented in the file `netcifar10.py`.

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


class Net_CIFAR10(netcore.Net_Core):
    def __init__(self, l1, l2, lr_mult, batch_size, epochs, k_folds, patience,
    optimizer, sgd_momentum):
        super(Net_CIFAR10, self).__init__(
            lr_mult=lr_mult,
            batch_size=batch_size,
            epochs=epochs,
            k_folds=k_folds,
            patience=patience,
            optimizer=optimizer,
            sgd_momentum=sgd_momentum,
        )
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, l1)
        self.fc2 = nn.Linear(l1, l2)
        self.fc3 = nn.Linear(l2, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

14.5.1 The Net_Core class

Net_CIFAR10 inherits from the class Net_Core which is implemented in the file netcore.py. It implements the additional attributes that are common to all neural network models. The Net_Core class is implemented in the file netcore.py. It implements hyperparameters as attributes, that are not used by the core_model, e.g.:

- optimizer (optimizer),
- learning rate (lr),
- batch size (batch_size),
- epochs (epochs),
- k_folds (k_folds), and
- early stopping criterion "patience" (patience).

Users can add further attributes to the class. The class Net_Core is shown below.

```
from torch import nn


class Net_Core(nn.Module):
    def __init__(self, lr_mult, batch_size, epochs, k_folds, patience,
        optimizer, sgd_momentum):
        super(Net_Core, self).__init__()
        self.lr_mult = lr_mult
        self.batch_size = batch_size
        self.epochs = epochs
        self.k_folds = k_folds
        self.patience = patience
        self.optimizer = optimizer
        self.sgd_momentum = sgd_momentum
```

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

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

spotPython's core_model implements an instance of the Net_CIFAR10 class. In addition to the basic neural network model, the core_model can use these additional attributes. spotPython provides methods for handling these additional attributes to guarantee 100% compatibility with the PyTorch classes. The method add_core_model_to_fun_control adds the hyperparameters and additional attributes to the fun_control dictionary. The method is shown below.

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

14.5.3 The Search Space: Hyperparameters

In Section 14.5.4, we first describe how to configure the search space with ray[tune] (as shown in PyTorch (2023a)) and then how to configure the search space with spotPython in -14.

14.5.4 Configuring the Search Space With Ray Tune

Ray Tune's search space can be configured as follows (PyTorch 2023a):

```
config = {
    "l1": tune.sample_from(lambda _: 2**np.random.randint(2, 9)),
    "l2": tune.sample_from(lambda _: 2**np.random.randint(2, 9)),
    "lr": tune.loguniform(1e-4, 1e-1),
    "batch_size": tune.choice([2, 4, 8, 16])
}
```

The tune.sample_from() function enables the user to define sample methods to obtain hyperparameters. In this example, the l1 and l2 parameters should be powers of 2 between 4 and 256, so either 4, 8, 16, 32, 64, 128, or 256. The lr (learning rate) should be uniformly sampled between 0.0001 and 0.1. Lastly, the batch size is a choice between 2, 4, 8, and 16.

At each trial, ray[tune] will randomly sample a combination of parameters from these search spaces. It will then train a number of models in parallel and find the best performing one among these. ray[tune] uses the ASHAScheduler which will terminate bad performing trials early.

14.5.5 Configuring the Search Space With spotPython

14.5.5.1 The `hyper_dict` Hyperparameters for the Selected Algorithm

`spotPython` uses `JSON` files for the specification of the hyperparameters. Users can specify their individual `JSON` files, or they can use the `JSON` files provided by `spotPython`. The `JSON` file for the `core_model` is called `torch_hyper_dict.json`.

In contrast to `ray[tune]`, `spotPython` can handle numerical, boolean, and categorical hyperparameters. They can be specified in the `JSON` file in a similar way as the numerical hyperparameters as shown below. Each entry in the `JSON` file represents one hyperparameter with the following structure: `type`, `default`, `transform`, `lower`, and `upper`.

```
"factor_hyperparameter": {
    "levels": ["A", "B", "C"],
    "type": "factor",
    "default": "B",
    "transform": "None",
    "core_model_parameter_type": "str",
    "lower": 0,
    "upper": 2},
```

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

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

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

```
 'default': 4,
 'transform': 'transform_power_2_int',
 'lower': 1,
 'upper': 4},
'epochs': {'type': 'int',
 'default': 3,
 'transform': 'transform_power_2_int',
 'lower': 3,
 'upper': 4},
'k_folds': {'type': 'int',
 'default': 1,
 'transform': 'None',
 'lower': 1,
 'upper': 1},
'patience': {'type': 'int',
 'default': 5,
 'transform': 'None',
 'lower': 2,
 'upper': 10},
'optimizer': {'levels': ['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}}
```

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

Ray tune (PyTorch 2023a) does not provide a way to change the specified hyperparameters without re-compilation. However, `spotPython` provides functions for modifying the hyperparameters, their bounds and factors as well as for activating and de-activating hyperparameters without re-compilation of the Python source code. These functions are described in the following.

14.6.0.1 Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model`

After specifying the model, the corresponding hyperparameters, their types and bounds are loaded from the `JSON` file `torch_hyper_dict.json`. After loading, the user can modify the hyperparameters, e.g., the bounds. `spotPython` provides a simple rule for de-activating hyperparameters: If the lower and the upper bound are set to identical values, the hyperparameter is de-activated. This is useful for the hyperparameter tuning, because it allows to specify a hyperparameter in the `JSON` file, but to de-activate it in the `fun_control` dictionary. This is done in the next step.

14.6.0.2 Modify Hyperparameters of Type numeric and integer (boolean)

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

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

14.6.0.3 Modify Hyperparameter of Type factor

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

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

```python
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"`.

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

As discussed in Section 14.6.1, there are some issues with the LBFGS optimizer. Therefore, the usage of the LBFGS optimizer is not deactivated in `spotPython` by default. However, the LBFGS optimizer can be activated by adding it to the list of optimizers. `Rprop` was removed, because it does perform very poorly (as some pre-tests have shown). However, it can also be activated by adding it to the list of optimizers. Since `SparseAdam` does not support dense gradients, `Adam` was used instead. Therefore, there are 10 default optimizers:

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

14.6.1 Optimizers

Table 14.1 shows some of the optimizers available in `PyTorch`:

a denotes (0.9,0.999), b (0.5,1.2), and c (1e-6, 50), respectively. R denotes `required, but unspecified`. "m" denotes `momentum`, "w_d" `weight_decay`, "d" `dampening`, "n" `nesterov`, "r" `rho`, "l_s" `learning rate for scaling delta`, "l_d" `lr_decay`, "b" `betas`, "l" `lambd`, "a" `alpha`, "m_d" for `momentum_decay`, "e" `etas`, and "s_s" for `step_sizes`.

Table 14.1: Optimizers available in PyTorch (selection). The default values are shown in the table.

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

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

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

14.7.1 Evaluation: Data Splitting

The evaluation procedure requires the specification of the way how the data is split into a train and a test set and the loss function (and a metric). As a default, `spotPython` provides a standard hold-out data split and cross validation.

14.7.2 Hold-out Data Split

If a hold-out data split is used, the data will be partitioned into a training, a validation, and a test data set. The split depends on the setting of the `eval` parameter. If `eval` is set to `train_hold_out`, one data set, usually the original training data set, is split into a new training and a validation data set. The training data set is used for training the model. The validation data set is used for the evaluation of the hyperparameter configuration and early stopping to prevent overfitting. In this case, the original test data set is not used.

> **Note**
>
> `spotPython` returns the hyperparameters of the machine learning and deep learning models, e.g., number of layers, learning rate, or optimizer, but not the model weights. Therefore, after the SPOT run is finished, the corresponding model with the optimized architecture has to be trained again with the best hyperparameter configuration. The training is performed on the training data set. The test data set is used for the final evaluation of the model.
> Summarizing, the following splits are performed in the hold-out setting:
>
> 1. Run `spotPython` with `eval` set to `train_hold_out` to determine the best hyperparameter configuration.
> 2. Train the model with the best hyperparameter configuration (“architecture”) on

the training data set: `train_tuned(model_spot, train, "model_spot.pt")`.
3. Test the model on the test data: `test_tuned(model_spot, test, "model_spot.pt")`

These steps will be exemplified in the following sections.

In addition to this `hold-out` setting, `spotPython` provides another hold-out setting, where an explicit test data is specified by the user that will be used as the validation set. To choose this option, the `eval` parameter is set to `test_hold_out`. In this case, the training data set is used for the model training. Then, the explicitly defined test data set is used for the evaluation of the hyperparameter configuration (the validation).

14.7.3 Cross-Validation

The cross validation setting is used by setting the `eval` parameter to `train_cv` or `test_cv`. In both cases, the data set is split into k folds. The model is trained on $k-1$ folds and evaluated on the remaining fold. This is repeated k times, so that each fold is used exactly once for evaluation. The final evaluation is performed on the test data set. The cross validation setting is useful for small data sets, because it allows to use all data for training and evaluation. However, it is computationally expensive, because the model has to be trained k times.

> **Note**
>
> Combinations of the above settings are possible, e.g., cross validation can be used for training and hold-out for evaluation or *vice versa*. Also, cross validation can be used for training and testing. Because cross validation is not used in the `PyTorch` tutorial (PyTorch 2023a), it is not considered further here.

14.7.4 Overview of the Evaluation Settings

14.7.4.1 Settings for the Hyperparameter Tuning

An overview of the training evaluations is shown in Table 14.2. `"train_cv"` and `"test_cv"` use `sklearn.model_selection.KFold()` internally. More details on the data splitting are provided in Section 22.14 (in the Appendix).

Table 14.2: Overview of the evaluation settings.

eval	train	test	function	comment
"train_hold_out"	✓		train_one_epoch(), validate_one_epoch() for early stopping	splits the train data set internally
"test_hold_out"	✓	✓	train_one_epoch(), validate_one_epoch() for early stopping	use the test data set for validate_one_epoch()
"train_cv"	✓		evaluate_cv(net, train)	CV using the train data set
"test_cv"		✓	evaluate_cv(net, test)	CV using the test data set . Identical to "train_cv", uses only test data.

14.7.4.2 Settings for the Final Evaluation of the Tuned Architecture

14.7.4.2.1 Training of the Tuned Architecture

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

14.7.4.2.2 Testing of the Tuned Architecture

`test_tuned(model, test)`: test the model on the test data set. No data splitting is performed. The (trained) model is evaluated using the `validate_fold_or_hold_out()` function. Note: During training, `"shuffle"` is set to `True`, whereas during testing, `"shuffle"` is set to `False`.

Section 22.14.1.4 describes the final evaluation of the tuned architecture.

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

14.7.5 Evaluation: Loss Functions and Metrics

The key "loss_function" specifies the loss function which is used during the optimization. There are several different loss functions under PyTorch's nn package. For example, a simple loss is MSELoss, which computes the mean-squared error between the output and the target. In this tutorial we will use CrossEntropyLoss, because it is also used in the PyTorch tutorial.

```
from torch.nn import CrossEntropyLoss
loss_function = CrossEntropyLoss()
fun_control.update({"loss_function": loss_function})
```

In addition to the loss functions, spotPython provides access to a large number of metrics.

- The key "metric_sklearn" is used for metrics that follow the scikit-learn conventions.
- The key "river_metric" is used for the river based evaluation (Montiel et al. 2021) via eval_oml_iter_progressive, and
- the key "metric_torch" is used for the metrics from TorchMetrics.

TorchMetrics is a collection of more than 90 PyTorch metrics, see https://torchmetrics.readthedocs.io/en/latest/. Because the PyTorch tutorial uses the accuracy as metric, we use the same metric here. Currently, accuracy is computed in the tutorial's example code. We will use TorchMetrics instead, because it offers more 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})
```

14.8 Step 8: Calling the SPOT Function

14.8.1 Preparing the SPOT Call

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

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

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

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

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

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

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

14.8.2 The Objective Function `fun_torch`

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

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

14.8.3 Using Default Hyperparameters or Results from Previous Runs

We add the default setting to the initial design:

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

14.8.4 Starting the Hyperparameter Tuning

The `spotPython` hyperparameter tuning is started by calling the `Spot` function. Here, we will run the tuner for approximately 30 minutes (`max_time`). Note: the initial design is always evaluated in the `spotPython` run. As a consequence, the run may take longer than specified by `max_time`, because the evaluation time of initial design (here: `init_size`, 10 points) is performed independently of `max_time`. During the run, results from the training is shown. These results can be visualized with Tensorboard as will be shown in Section 14.9.

```
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, 'p
Epoch: 1 |

MulticlassAccuracy: 0.3641000092029572 | Loss: 1.6939691730499267 | Acc: 0.3641000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.4142999947071075 | Loss: 1.5814018835067749 | Acc: 0.4143000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.4537000060081482 | Loss: 1.4788755718231201 | Acc: 0.4537000000000000.
Epoch: 4 |
```

```
MulticlassAccuracy: 0.4904499948024750 | Loss: 1.3833515543937682 | Acc: 0.4904500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5210499763488770 | Loss: 1.3204596006393432 | Acc: 0.5210500000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5243499875068665 | Loss: 1.3099725577354431 | Acc: 0.5243500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5330500006675720 | Loss: 1.2798596569061280 | Acc: 0.5330500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5490999817848206 | Loss: 1.2539205450057984 | Acc: 0.5491000000000000.
Epoch: 9 |

MulticlassAccuracy: 0.5559499859809875 | Loss: 1.2304777290344238 | Acc: 0.5559500000000001.
Epoch: 10 |

MulticlassAccuracy: 0.5624499917030334 | Loss: 1.2237399903297423 | Acc: 0.5624500000000000.
Epoch: 11 |

MulticlassAccuracy: 0.5702000260353088 | Loss: 1.1944132906913758 | Acc: 0.5702000000000000.
Epoch: 12 |

MulticlassAccuracy: 0.5759000182151794 | Loss: 1.1969344672203064 | Acc: 0.5759000000000000.
Epoch: 13 |

MulticlassAccuracy: 0.5732499957084656 | Loss: 1.1987279416084289 | Acc: 0.5732500000000000.
Epoch: 14 |

MulticlassAccuracy: 0.5793499946594238 | Loss: 1.1867230147361756 | Acc: 0.5793500000000000.
Epoch: 15 |

MulticlassAccuracy: 0.5842000246047974 | Loss: 1.1813372245788574 | Acc: 0.5842000000000001.
Epoch: 16 |
```

```
MulticlassAccuracy: 0.5970000028610229 | Loss: 1.1612035545349122 | Acc: 0.5970000000000000.
Returned to Spot: Validation loss: 1.1612035545349122

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

MulticlassAccuracy: 0.4357500076293945 | Loss: 1.5316598705530167 | Acc: 0.4357500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.4796999990940094 | Loss: 1.4239955782413483 | Acc: 0.4797000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.4805000126361847 | Loss: 1.4398172856092453 | Acc: 0.4805000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5128499865531921 | Loss: 1.3537848952770233 | Acc: 0.5128500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5312500000000000 | Loss: 1.3077760408997536 | Acc: 0.5312500000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5289999842643738 | Loss: 1.3229600148439407 | Acc: 0.5290000000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5407999753952026 | Loss: 1.2901836607098580 | Acc: 0.5407999999999999.
Epoch: 8 |

MulticlassAccuracy: 0.5236999988555908 | Loss: 1.3462823650717735 | Acc: 0.5237000000000001.
Returned to Spot: Validation loss: 1.3462823650717735

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

MulticlassAccuracy: 0.1509999930858612 | Loss: 2.2169039542436599 | Acc: 0.1510000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.1518500000238419 | Loss: 2.3466526296603960 | Acc: 0.1518500000000000.
Epoch: 3 |
```

```
MulticlassAccuracy: 0.1010499969124794 | Loss: 2.3158330991029739 | Acc: 0.1010500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.1024999991059303 | Loss: 2.3079263017892839 | Acc: 0.1025000000000000.
Early stopping at epoch 3
Returned to Spot: Validation loss: 2.307926301789284

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

MulticlassAccuracy: 0.4180999994277954 | Loss: 1.5741076424717904 | Acc: 0.4181000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.4815999865531921 | Loss: 1.4420298090338708 | Acc: 0.4816000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.5117999911308289 | Loss: 1.3745080009639263 | Acc: 0.5118000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5280500054359436 | Loss: 1.3403449234619738 | Acc: 0.5280500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5400500297546387 | Loss: 1.2988252326935530 | Acc: 0.5400500000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5516499876976013 | Loss: 1.2786387486442923 | Acc: 0.5516500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5594999790191650 | Loss: 1.2782457950979471 | Acc: 0.5595000000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5744500160217285 | Loss: 1.2365814841069280 | Acc: 0.5744500000000000.
Returned to Spot: Validation loss: 1.236581484106928

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

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

```
MulticlassAccuracy: 0.4896000027656555 | Loss: 1.3984203063964844 | Acc: 0.4896000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.5349000096321106 | Loss: 1.2954318794250488 | Acc: 0.5349000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.5626000165939331 | Loss: 1.2502731824398041 | Acc: 0.5626000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5770999789237976 | Loss: 1.2409635678052902 | Acc: 0.5770999999999999.
Epoch: 5 |

MulticlassAccuracy: 0.5669999718666077 | Loss: 1.2937656073570252 | Acc: 0.5669999999999999.
Epoch: 6 |

MulticlassAccuracy: 0.5769500136375427 | Loss: 1.2938319791793824 | Acc: 0.5769500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5764999985694885 | Loss: 1.3551661030650139 | Acc: 0.5765000000000000.
Early stopping at epoch 6
Returned to Spot: Validation loss: 1.355166103065014

spotPython tuning: 1.1612035545349122 [##--------] 17.10%


config: {'l1': 64, 'l2': 4, 'lr_mult': 1.0, 'batch_size': 32, 'epochs': 16, 'k_folds': 0, 'pa
Epoch: 1 |

MulticlassAccuracy: 0.2596000134944916 | Loss: 1.8270551212310790 | Acc: 0.2596000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.2905499935150146 | Loss: 1.7429899692535400 | Acc: 0.2905500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.2963500022888184 | Loss: 1.6904951917648317 | Acc: 0.2963500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.3345000147819519 | Loss: 1.6367128517150880 | Acc: 0.3345000000000000.
Epoch: 5 |
```

```
MulticlassAccuracy: 0.3504999876022339 | Loss: 1.6165647289276124 | Acc: 0.3505000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.3634499907493591 | Loss: 1.6153322587966918 | Acc: 0.3634500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.3957000076770782 | Loss: 1.5601136526107788 | Acc: 0.3957000000000000.
Epoch: 8 |

MulticlassAccuracy: 0.4075999855995178 | Loss: 1.5410722332000732 | Acc: 0.4076000000000000.
Epoch: 9 |

MulticlassAccuracy: 0.4134500026702881 | Loss: 1.5324218894958497 | Acc: 0.4134500000000000.
Epoch: 10 |

MulticlassAccuracy: 0.4305999875068665 | Loss: 1.5009049962997436 | Acc: 0.4306000000000000.
Epoch: 11 |

MulticlassAccuracy: 0.4365999996662140 | Loss: 1.5116015024185181 | Acc: 0.4366000000000000.
Epoch: 12 |

MulticlassAccuracy: 0.4506500065326691 | Loss: 1.4793368362426758 | Acc: 0.4506500000000000.
Epoch: 13 |

MulticlassAccuracy: 0.4398500025272369 | Loss: 1.4825679677963257 | Acc: 0.4398500000000000.
Epoch: 14 |

MulticlassAccuracy: 0.4657999873161316 | Loss: 1.4701670835494995 | Acc: 0.4658000000000000.
Epoch: 15 |

MulticlassAccuracy: 0.4758999943733215 | Loss: 1.4682736368179321 | Acc: 0.4759000000000000.
Epoch: 16 |

MulticlassAccuracy: 0.4839999973773956 | Loss: 1.4360087179183960 | Acc: 0.4840000000000000.
Returned to Spot: Validation loss: 1.436008717918396
spotPython tuning: 1.1612035545349122 [#####-----] 45.63%


config: {'l1': 16, 'l2': 32, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Epoch: 1 |
```

```
MulticlassAccuracy: 0.4415999948978424 | Loss: 1.5234199389010668 | Acc: 0.4416000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.4919500052928925 | Loss: 1.4223259533450008 | Acc: 0.4919500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.5443999767303467 | Loss: 1.2933703027129173 | Acc: 0.5444000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.5418999791145325 | Loss: 1.3234680540699513 | Acc: 0.5419000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5565500259399414 | Loss: 1.2811511654891075 | Acc: 0.5565500000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5722500085830688 | Loss: 1.2504825864769518 | Acc: 0.5722500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5753999948501587 | Loss: 1.2298460423439741 | Acc: 0.5754000000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5842499732971191 | Loss: 1.2259684566644953 | Acc: 0.5842500000000000.
Returned to Spot: Validation loss: 1.2259684566644953

spotPython tuning: 1.1612035545349122 [######----] 61.53%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 32, 'epochs': 16, 'k_folds': 0, 'p
Epoch: 1 |

MulticlassAccuracy: 0.3531999886035919 | Loss: 1.7122918035507202 | Acc: 0.3532000000000000.
Epoch: 2 |

MulticlassAccuracy: 0.4324499964714050 | Loss: 1.5243975790023803 | Acc: 0.4324500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.4744000136852264 | Loss: 1.4572544511795045 | Acc: 0.4744000000000000.
Epoch: 4 |
```

```
MulticlassAccuracy: 0.5116500258445740 | Loss: 1.3741200543403624 | Acc: 0.5116500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.5299999713897705 | Loss: 1.3390167931556702 | Acc: 0.5300000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.5336999893188477 | Loss: 1.3258737889289856 | Acc: 0.5337000000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5511000156402588 | Loss: 1.2846284455299377 | Acc: 0.5511000000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5511999726295471 | Loss: 1.2985027008056640 | Acc: 0.5512000000000000.
Epoch: 9 |

MulticlassAccuracy: 0.5548999905586243 | Loss: 1.2982149655342101 | Acc: 0.5548999999999999.
Epoch: 10 |

MulticlassAccuracy: 0.5655000209808350 | Loss: 1.2660368131637574 | Acc: 0.5655000000000000.
Epoch: 11 |

MulticlassAccuracy: 0.5680000185966492 | Loss: 1.2821598432540893 | Acc: 0.5679999999999999.
Epoch: 12 |

MulticlassAccuracy: 0.5708000063896179 | Loss: 1.2865115547180175 | Acc: 0.5708000000000000.
Epoch: 13 |

MulticlassAccuracy: 0.5654500126838684 | Loss: 1.3399226131916047 | Acc: 0.5654500000000000.
Early stopping at epoch 12
Returned to Spot: Validation loss: 1.3399226131916047
spotPython tuning: 1.1612035545349122 [#########-] 85.36%


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

MulticlassAccuracy: 0.2322999984025955 | Loss: 1.9758470226287841 | Acc: 0.2323000000000000.
Epoch: 2 |
```

```
MulticlassAccuracy: 0.2743000090122223 | Loss: 1.8700803245544433 | Acc: 0.2743000000000000.
Epoch: 3 |

MulticlassAccuracy: 0.2345000058412552 | Loss: 1.9918049844741821 | Acc: 0.2345000000000000.
Epoch: 4 |

MulticlassAccuracy: 0.2197500020265579 | Loss: 2.0200491023063658 | Acc: 0.2197500000000000.
Epoch: 5 |

MulticlassAccuracy: 0.2196999937295914 | Loss: 2.0912528437614442 | Acc: 0.2197000000000000.
Early stopping at epoch 4
Returned to Spot: Validation loss: 2.091252843761444
spotPython tuning: 1.1612035545349122 [#########-] 92.93%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 92.97%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 93.01%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 93.05%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 93.09%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 93.14%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 93.18%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 93.22%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 93.26%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 93.30%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 93.34%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 93.39%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 93.43%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 93.47%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 93.51%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 93.55%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 93.59%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 93.63%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 93.67%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 93.71%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 93.75%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 93.79%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 93.83%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 93.87%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 93.91%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 93.95%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 94.00%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 94.04%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 94.08%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 94.12%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 94.16%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 94.20%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 94.24%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 94.28%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 94.32%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa

Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 94.37%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa

Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 94.41%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa

Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 94.46%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 94.50%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 94.54%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 94.58%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 94.62%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 94.66%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 94.71%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 94.75%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 94.79%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 94.83%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 94.88%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 94.92%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [#########-] 94.96%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 95.01%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 95.05%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 95.09%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 95.14%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 95.18%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 95.22%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 95.27%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 95.31%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 95.35%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 95.40%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 95.44%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 95.49%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 95.53%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 95.58%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 95.62%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 95.67%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 95.71%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 95.75%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 95.80%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 95.84%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 95.89%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 95.93%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 95.97%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 96.02%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 96.06%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 96.10%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 96.15%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 96.19%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 96.23%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 96.28%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa

Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 96.32%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa

Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 96.37%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 96.41%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 96.45%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 96.49%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 96.53%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 96.58%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa

Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 96.62%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 96.66%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 96.71%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa

Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 96.75%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 96.79%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 96.84%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa

Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 96.88%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 96.92%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 96.96%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa

Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 97.01%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 97.05%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 97.10%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 97.14%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 97.18%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 97.23%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa

Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 97.27%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 97.31%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 97.36%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa

Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 97.40%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa

Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 97.45%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa



Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 97.49%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 97.53%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 97.58%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 97.62%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 97.66%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 97.70%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 97.75%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 97.79%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 97.83%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 97.88%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 97.92%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 97.96%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa

Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 98.01%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 98.05%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa

Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 98.09%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 98.14%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 98.18%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 98.22%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 98.26%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 98.31%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 98.35%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 98.39%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 98.43%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 98.48%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 98.52%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 98.56%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 98.60%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 98.64%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 98.68%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 98.73%


config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 98.77%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 98.81%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa

Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 98.86%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 98.90%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa

Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 98.94%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 98.98%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 99.03%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 99.07%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 99.11%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 99.15%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 99.19%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 99.23%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 99.27%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 99.31%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 99.36%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 99.40%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 99.44%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 99.48%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 99.52%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 99.56%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 99.61%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 99.65%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 99.69%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 99.73%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 99.77%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 99.82%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 99.86%

config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 99.91%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 99.95%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 99.99%
```

```
config: {'l1': 256, 'l2': 8, 'lr_mult': 1.0, 'batch_size': 4, 'epochs': 8, 'k_folds': 0, 'pa
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adagrad.__init__() got
Returned to Spot: Validation loss: nan
spotPython tuning: 1.1612035545349122 [##########] 100.00% Done...
```

```
<spotPython.spot.spot.Spot at 0x2b847b940>
```

14.9 Step 9: Tensorboard

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

14.9.1 Tensorboard: Start Tensorboard

Start TensorBoard through the command line to visualize data you logged. Specify the root log directory as used in `fun_control = fun_control_init(task="regression", tensorboard_path="runs/24_spot_torch_regression")` as the `tensorboard_path`. The argument logdir points to directory where TensorBoard will look to find event files that it can display. TensorBoard will recursively walk the directory structure rooted at logdir, looking for *.tfevents.* files.

```
tensorboard --logdir=runs
```

Go to the URL it provides or to http://localhost:6006/. The following figures show some screenshots of Tensorboard.

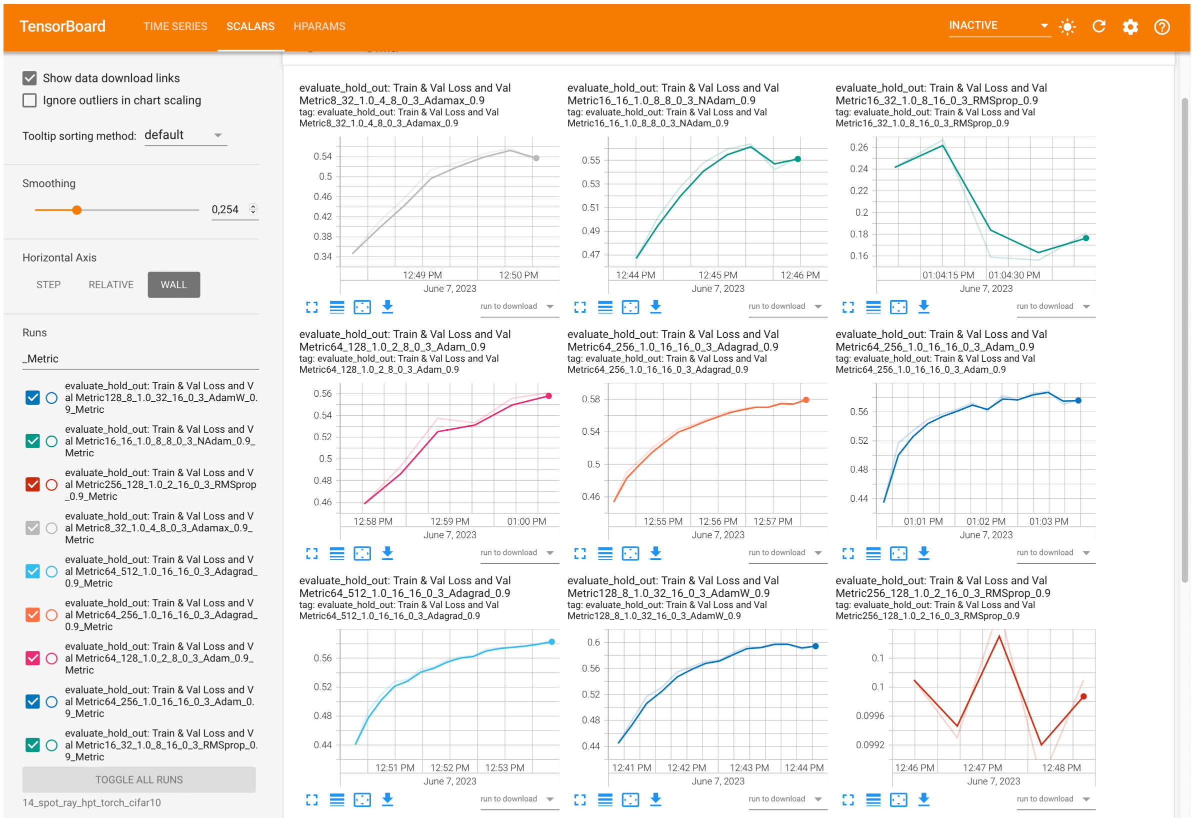


Figure 14.1: Tensorboard

Trial ID	Show Metrics	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 14.2: Tensorboard

14.9.2 Saving the State of the Notebook

The state of the notebook can be saved and reloaded as follows:

```
import pickle
SAVE = False
LOAD = False

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

if LOAD:
    result_file_name = "add_the_name_of_the_result_file_here.pkl"
    with open(result_file_name, 'rb') as f:
        spot_tuner =  pickle.load(f)
```

14.10 Step 10: Results

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

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

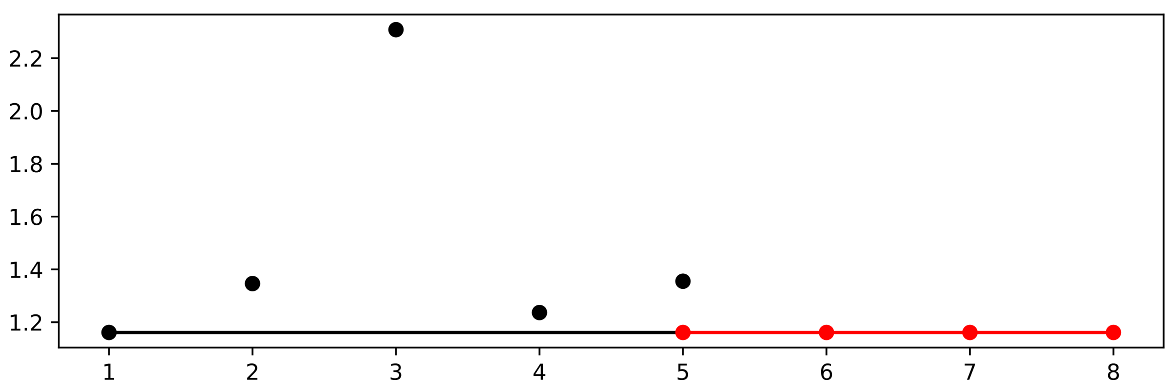


Figure 14.3: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

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

In addition to the improved ("optimized") hyperparameter values, `spotPython` allows a statistical analysis, e.g., a sensitivity analysis, of the results. We can print the results of the hyperparameter tuning, see **?@tbl-results**. The table shows the hyperparameters, their types, default values, lower and upper bounds, and the transformation function. The column "tuned" shows the tuned values. The column "importance" shows the importance of the hyperparameters. The column "stars" shows the importance of the hyperparameters in stars. The importance is computed by the SPOT software.

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

```
| name         | type   | default   |   lower |   upper |   tuned | transform              |
|--------------|--------|-----------|---------|---------|---------|------------------------|--
| l1           | int    | 5         |     2.0 |     9.0 |     7.0 | transform_power_2_int |
| l2           | int    | 5         |     2.0 |     9.0 |     3.0 | transform_power_2_int |
| lr_mult      | float  | 1.0       |     1.0 |     1.0 |     1.0 | None                   |
| batch_size   | int    | 4         |     1.0 |     5.0 |     5.0 | transform_power_2_int |
| epochs       | int    | 3         |     3.0 |     4.0 |     4.0 | transform_power_2_int |
| k_folds      | int    | 1         |     0.0 |     0.0 |     0.0 | None                   |
| patience     | int    | 5         |     3.0 |     3.0 |     3.0 | None                   |
| optimizer    | factor | SGD       |     0.0 |     9.0 |     3.0 | None                   |
| sgd_momentum | float  | 0.0       |     0.9 |     0.9 |     0.9 | None                   |
```

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

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

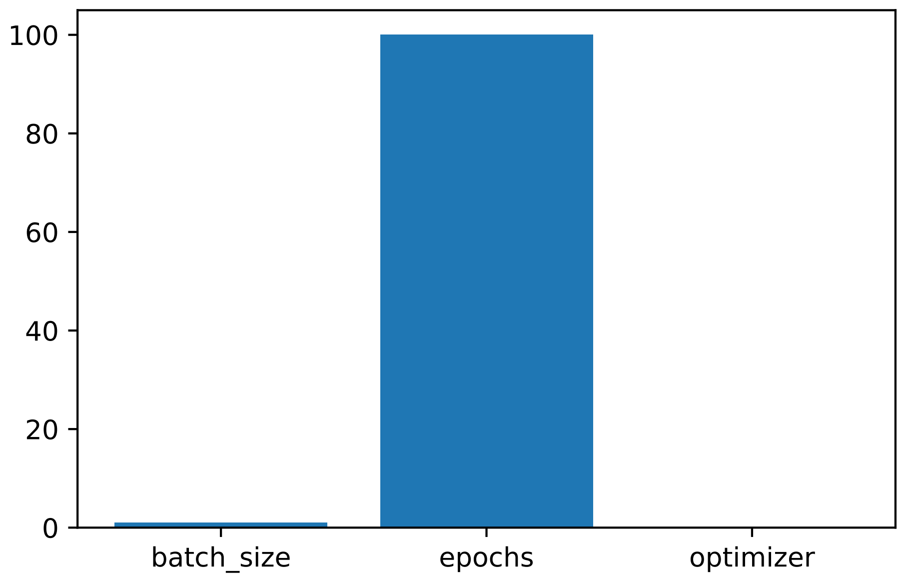


Figure 14.4: Variable importance plot, threshold 0.025.

14.10.1 Get the Tuned Architecture (SPOT Results)

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

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

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

14.10.2 Get Default Hyperparameters

In a similar manner as in Section 14.10.1, the default hyperparameters can be obtained.

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

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

14.10.3 Evaluation of the Default Architecture

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

This evaluation is similar to the final evaluation in PyTorch (2023a).

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

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

```
Epoch: 1 |

MulticlassAccuracy: 0.1132500022649765 | Loss: 2.3089407604217529 | Acc: 0.1132500000000000.
Epoch: 2 |

MulticlassAccuracy: 0.1303499937057495 | Loss: 2.3055755001068117 | Acc: 0.1303500000000000.
Epoch: 3 |

MulticlassAccuracy: 0.1338499933481216 | Loss: 2.3028430467605592 | Acc: 0.1338500000000000.
Epoch: 4 |

MulticlassAccuracy: 0.1358000040054321 | Loss: 2.2997729673385621 | Acc: 0.1358000000000000.
Epoch: 5 |

MulticlassAccuracy: 0.1513999998569489 | Loss: 2.2953105113983154 | Acc: 0.1514000000000000.
Epoch: 6 |

MulticlassAccuracy: 0.1662999987602234 | Loss: 2.2881209154129030 | Acc: 0.1663000000000000.
Epoch: 7 |

MulticlassAccuracy: 0.1784500032663345 | Loss: 2.2723436662673948 | Acc: 0.1784500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.1773499995470047 | Loss: 2.2327963512420652 | Acc: 0.1773500000000000.
Returned to Spot: Validation loss: 2.232796351242065

MulticlassAccuracy: 0.1791000068187714 | Loss: 2.2344608154296877 | Acc: 0.1791000000000000.
Final evaluation: Validation loss: 2.2344608154296877
Final evaluation: Validation metric: 0.17910000681877136
----------------------------------------------

(2.2344608154296877, nan, tensor(0.1791))
```

14.10.4 Evaluation of the Tuned Architecture

The following code trains the model `model_spot`.

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

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

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

```
Epoch: 1 |


MulticlassAccuracy: 0.3605499863624573 | Loss: 1.6898259429931641 | Acc: 0.3605500000000000.
Epoch: 2 |


MulticlassAccuracy: 0.4447999894618988 | Loss: 1.4984460794448853 | Acc: 0.4448000000000000.
Epoch: 3 |


MulticlassAccuracy: 0.4709999859333038 | Loss: 1.4622202505111694 | Acc: 0.4710000000000000.
Epoch: 4 |


MulticlassAccuracy: 0.5158500075340271 | Loss: 1.3340031201362610 | Acc: 0.5158500000000000.
Epoch: 5 |


MulticlassAccuracy: 0.5267500281333923 | Loss: 1.3007940133094789 | Acc: 0.5267500000000001.
Epoch: 6 |
```

```
MulticlassAccuracy: 0.5456500053405762 | Loss: 1.2556657067298889 | Acc: 0.5456500000000000.
Epoch: 7 |

MulticlassAccuracy: 0.5636500120162964 | Loss: 1.2122078575134276 | Acc: 0.5636500000000000.
Epoch: 8 |

MulticlassAccuracy: 0.5594000220298767 | Loss: 1.2333222242355346 | Acc: 0.5594000000000000.
Epoch: 9 |

MulticlassAccuracy: 0.5665000081062317 | Loss: 1.2116268886566162 | Acc: 0.5665000000000000.
Epoch: 10 |

MulticlassAccuracy: 0.5669500231742859 | Loss: 1.2108266097068787 | Acc: 0.5669500000000000.
Epoch: 11 |

MulticlassAccuracy: 0.5777999758720398 | Loss: 1.1892622032165527 | Acc: 0.5778000000000000.
Epoch: 12 |

MulticlassAccuracy: 0.5852000117301941 | Loss: 1.1775404059410095 | Acc: 0.5852000000000001.
Epoch: 13 |

MulticlassAccuracy: 0.5956000089645386 | Loss: 1.1646465710639953 | Acc: 0.5956000000000000.
Epoch: 14 |

MulticlassAccuracy: 0.5853499770164490 | Loss: 1.2187978922843934 | Acc: 0.5853500000000000.
Epoch: 15 |

MulticlassAccuracy: 0.6003500223159790 | Loss: 1.1737062762260437 | Acc: 0.6003500000000001.
Epoch: 16 |

MulticlassAccuracy: 0.5947499871253967 | Loss: 1.1896330574035645 | Acc: 0.5947500000000000.
Early stopping at epoch 15
Returned to Spot: Validation loss: 1.1896330574035645

MulticlassAccuracy: 0.6001999974250793 | Loss: 1.1962783707978246 | Acc: 0.6002000000000000.
Final evaluation: Validation loss: 1.1962783707978246
Final evaluation: Validation metric: 0.6001999974250793
----------------------------------------------

(1.1962783707978246, nan, tensor(0.6002))
```

14.10.5 Detailed Hyperparameter Plots

The contour plots in this section visualize the interactions of the three most important hyperparameters. Since some of these hyperparameters take 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)
```

```
batch_size:  1.0151632482140391
epochs:  100.0
optimizer:  0.04753957677412226
```

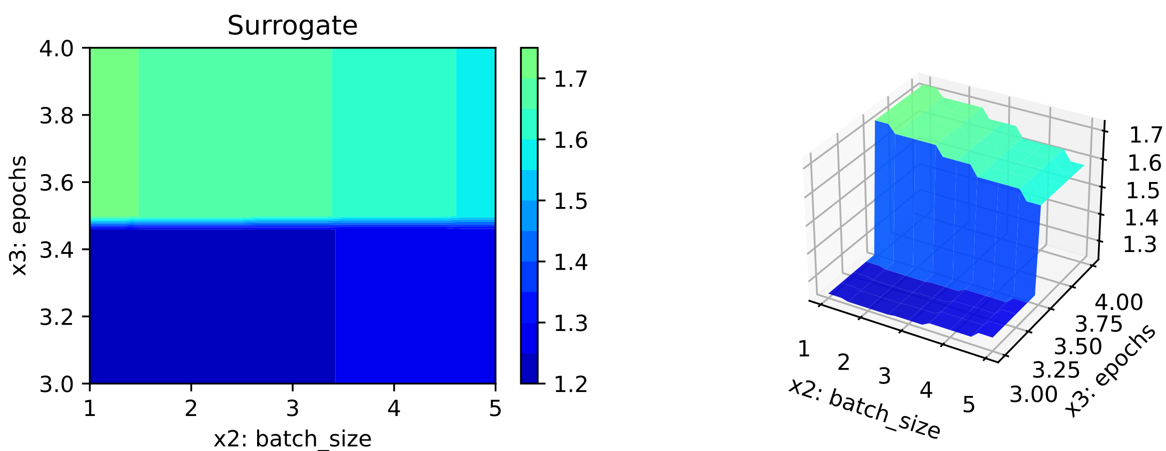
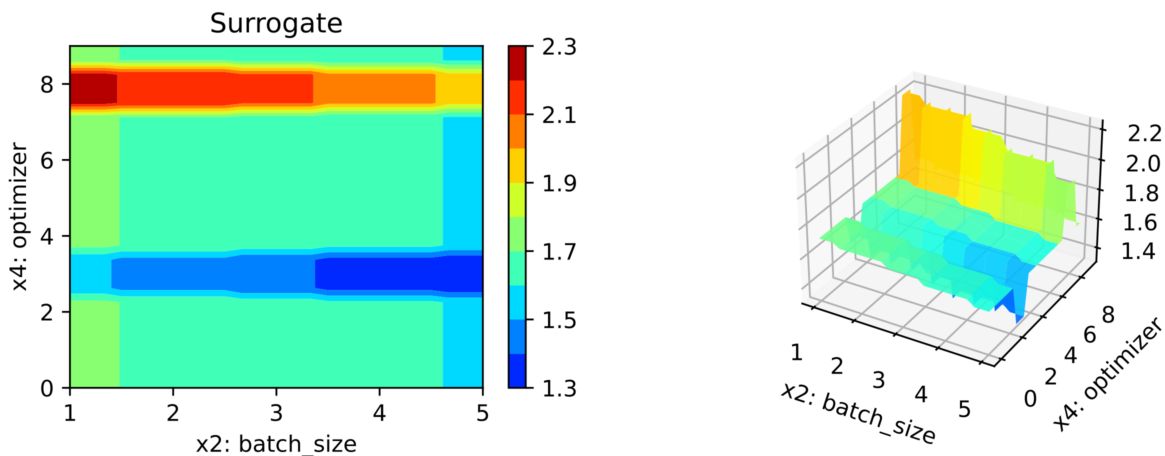
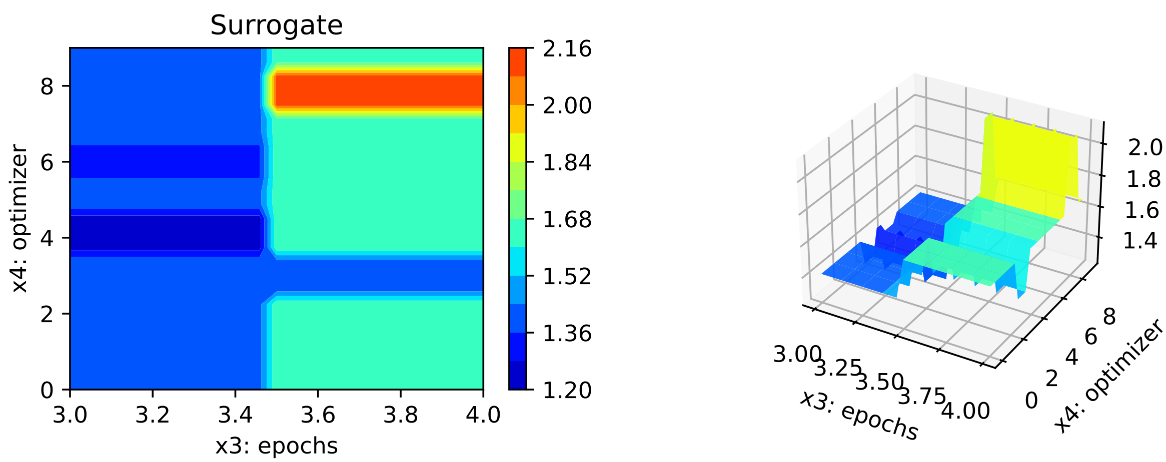


Figure 14.5: Contour plots.





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

```
spot_tuner.parallel_plot()
```

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

Parallel coordinates plots

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

14.11 Summary and Outlook

This tutorial presents the hyperparameter tuning open source software `spotPython` for `PyTorch`. To show its basic features, a comparison with the "official" `PyTorch` hyperparameter tuning tutorial (PyTorch 2023a) is presented. Some of the advantages of `spotPython` are:

- Numerical and categorical hyperparameters.
- Powerful surrogate models.
- Flexible approach and easy to use.
- Simple JSON files for the specification of the hyperparameters.
- Extension of default and user specified network classes.
- Noise handling techniques.
- Interaction with `tensorboard`.

Currently, only rudimentary parallel and distributed neural network training is possible, but these capabilities will be extended in the future. The next version of `spotPython` will also include a more detailed documentation and more examples.

> **! Important**
>
> Important: This tutorial does not present a complete benchmarking study (Bartz-Beielstein et al. 2020). The results are only preliminary and highly dependent on the local configuration (hard- and software). Our goal is to provide a first impression of the performance of the hyperparameter tuning package `spotPython`. To demonstrate its capabilities, a quick comparison with `ray[tune]` was performed. `ray[tune]` was chosen, because it is presented as "an industry standard tool for distributed hyperparameter tuning." The results should be interpreted with care.

14.12 Appendix

14.12.1 Sample Output From Ray Tune's Run

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

```
Number of trials: 10 (10 TERMINATED)
------+------+-------------+--------------+---------+------------+--------------------+
|   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
```

15 HPT: sklearn RandomForestClassifier VBDP Data

This document refers to the following software versions:

- `python`: 3.10.10

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

```
spotPython                    0.2.46
spotRiver                     0.0.93
```

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

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

```
!pip install spotPython
```

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

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

15.1 Step 1: Setup

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

```
MAX_TIME = 1
INIT_SIZE = 5
ORIGINAL = False
```

```
import os
import copy
import socket
from datetime import datetime
from dateutil.tz import tzlocal
start_time = datetime.now(tzlocal())
HOSTNAME = socket.gethostname().split(".")[0]
experiment_name = '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_2023-06-27_03-50-59
```

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

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

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

```
from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="classification",
    tensorboard_path="runs/16_spot_hpt_sklearn_classification")
```

15.3 Step 3: PyTorch Data Loading

15.3.1 Load Data: Classification VBDP

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

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

```
(707, 65)
```

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	...	x56	x57	x58	x59	x60	x61	x62	x63
0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	...	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0
3	0.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	0.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0

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

15.3.2 Holdout Train and Test Data

We split out a hold-out test set (25% of the data) so we can calculate an example MAP@K

```python
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

```python
# add the dataset to the fun_control
fun_control.update({"data": train_df, # full dataset,
               "train": train,
               "test": test,
               "n_samples": n_samples,
               "target_column": target_column})
```

15.4 Step 4: Specification of the Preprocessing Model

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

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

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

```
# prep_model = StandardScaler()
# fun_control.update({"prep_model": prep_model})
```

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

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

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

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

`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
```

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

15.6.1 Modify hyperparameter of type numeric and integer (boolean)

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

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

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

15.6.2 Modify hyperparameter of type factor

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

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

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

The new setting can be controlled via:

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

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

> **Note: RandomForestClassifier and Out-of-bag Estimation**
>
> Since `oob_score` requires the `bootstrap` hyperparameter to `True`, we set the `oob_score` parameter to `False`. The `oob_score` is later discussed in Section 15.7.3.

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

15.6.3 Optimizers

Optimizers are described in Section 14.6.1.

15.6.4 Selection of the Objective: Metric and Loss Functions

- Machine learning models are optimized with respect to a metric, for example, the `accuracy` function.
- Deep learning, e.g., neural networks are optimized with respect to a loss function, for example, the `cross_entropy` function and evaluated with respect to a metric, for example, the `accuracy` function.

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

The loss function, that is usually used in deep learning for optimizing the weights of the net, is stored in the `fun_control` dictionary as `"loss_function"`.

15.7.1 Metric Function

There are two different types of metrics in `spotPython`:

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

We will consider multi-class classification metrics, e.g., `mapk_score` and `top_k_accuracy_score`.

> **Predict Probabilities**
>
> In this multi-class classification example the machine learning algorithm should return the probabilities of the specific classes (`"predict_proba"`) instead of the predicted values.

We set `"predict_proba"` to `True` in the `fun_control` dictionary.

15.7.1.1 The MAPK Metric

To select the MAPK metric, the following two entries can be added to the `fun_control` dictionary:

`"metric_sklearn": mapk_score"`

`"metric_params": {"k": 3}`.

15.7.1.2 Other Metrics

Alternatively, other metrics for multi-class classification can be used, e.g.,: * top_k_accuracy_score or * roc_auc_score

The metric `roc_auc_score` requires the parameter `"multi_class"`, e.g.,

`"multi_class": "ovr"`.

This is set in the `fun_control` dictionary.

> **Weights**
>
> `spotPython` performs a minimization, therefore, metrics that should be maximized have to be multiplied by -1. This is done by setting `"weights"` to `-1`.

- The complete setup for the metric in our example is:

```python
from spotPython.utils.metrics import mapk_score
fun_control.update({
               "weights": -1,
               "metric_sklearn": mapk_score,
               "predict_proba": True,
               "metric_params": {"k": 3},
               })
```

15.7.2 Evaluation on Hold-out Data

- The default method for computing the performance is `"eval_holdout"`.
- Alternatively, cross-validation can be used for every machine learning model.
- Specifically for RandomForests, the OOB-score can be used.

```
fun_control.update({
    "eval": "train_hold_out",
})
```

15.7.3 OOB Score

Using the OOB-Score is a very efficient way to estimate the performance of a random forest classifier. The OOB-Score is calculated on the training data and does not require a hold-out test set. If the OOB-Score is used, the key "eval" in the `fun_control` dictionary should be set to `"oob_score"` as shown below.

> **OOB-Score**
>
> In addition to setting the key `"eval"` in the `fun_control` dictionary to `"oob_score"`, the keys `"oob_score"` and `"bootstrap"` have to be set to `True`, because the OOB-Score requires the bootstrap method.

- Uncomment the following lines to use the OOB-Score:

```
fun_control.update({
    "eval": "eval_oob_score",
})
fun_control = modify_hyper_parameter_bounds(fun_control, "bootstrap", bounds=[1, 1])
fun_control = modify_hyper_parameter_bounds(fun_control, "oob_score", bounds=[1, 1])
```

15.7.3.1 Cross Validation

Instead of using the OOB-score, the classical cross validation can be used. The number of folds is set by the key `"k_folds"`. For example, to use 5-fold cross validation, the key `"k_folds"` is set to `5`. Uncomment the following line to use cross validation:

```
# fun_control.update({
#      "eval": "train_cv",
#      "k_folds": 10,
# })
```

15.8 Step 8: Calling the SPOT Function

15.8.1 Preparing the SPOT Call

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

```python
# 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")
```

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

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

15.8.2 The Objective Function

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

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

15.8.3 Run the `Spot` Optimizer

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

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

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

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

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

```
spotPython tuning: -0.32987421383647797 [----------] 0.36%

spotPython tuning: -0.32987421383647797 [----------] 0.89%

spotPython tuning: -0.3525157232704403 [----------] 1.32%

spotPython tuning: -0.3525157232704403 [----------] 1.90%

spotPython tuning: -0.3525157232704403 [----------] 2.86%

spotPython tuning: -0.3525157232704403 [----------] 3.57%

spotPython tuning: -0.35817610062893085 [#---------] 5.01%

spotPython tuning: -0.35817610062893085 [#---------] 6.09%

spotPython tuning: -0.35817610062893085 [#---------] 7.95%

spotPython tuning: -0.35817610062893085 [#---------] 9.82%

spotPython tuning: -0.35817610062893085 [#---------] 12.14%

spotPython tuning: -0.35817610062893085 [#---------] 13.92%

spotPython tuning: -0.3613207547169811 [##--------] 17.06%

spotPython tuning: -0.3613207547169811 [##--------] 21.32%

spotPython tuning: -0.3613207547169811 [##--------] 24.89%
```

```
spotPython tuning: -0.3613207547169811 [###-------] 28.78%

spotPython tuning: -0.3613207547169811 [###-------] 32.33%

spotPython tuning: -0.3613207547169811 [####------] 35.61%

spotPython tuning: -0.36761006289308173 [####------] 39.11%

spotPython tuning: -0.36761006289308173 [####------] 44.31%

spotPython tuning: -0.36761006289308173 [#####-----] 48.60%

spotPython tuning: -0.36761006289308173 [#####-----] 52.08%

spotPython tuning: -0.36761006289308173 [######----] 55.89%

spotPython tuning: -0.36761006289308173 [######----] 59.82%

spotPython tuning: -0.36761006289308173 [######----] 63.40%

spotPython tuning: -0.36761006289308173 [#######---] 66.81%

spotPython tuning: -0.36761006289308173 [#######---] 70.62%

spotPython tuning: -0.36761006289308173 [########--] 75.61%

spotPython tuning: -0.36761006289308173 [########--] 80.19%

spotPython tuning: -0.36761006289308173 [#########-] 85.77%

spotPython tuning: -0.36761006289308173 [#########-] 91.03%

spotPython tuning: -0.36761006289308173 [##########] 96.97%

spotPython tuning: -0.36761006289308173 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x13d71feb0>
```

15.9 Step 9: Tensorboard

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

15.10 Step 10: Results

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

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

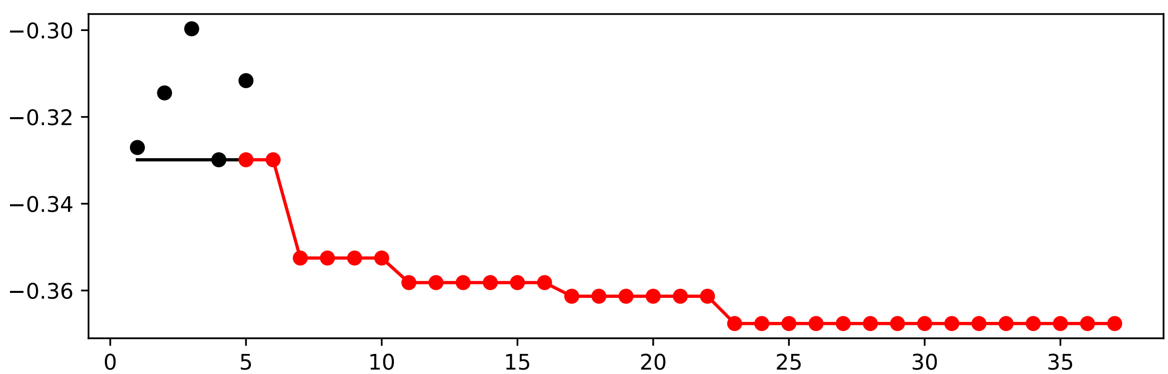


Figure 15.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- Print the results

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

```
| name                      | type   | default   |   lower |   upper |                  tuned
|---------------------------|--------|-----------|---------|---------|-----------------------
| n_estimators              | int    | 7         |     5.0 |    10.0 |                    9.0
| criterion                 | factor | gini      |     0.0 |     2.0 |                    0.0
| max_depth                 | int    | 10        |     1.0 |    20.0 |                   11.0
| min_samples_split         | int    | 2         |     2.0 |   100.0 |                   19.0
| min_samples_leaf          | int    | 1         |     1.0 |    25.0 |                    1.0
```

```
| min_weight_fraction_leaf | float  | 0.0       |      0.0 |     0.01 | 0.0037800008096200988
| max_features             | factor | sqrt      |      0.0 |      1.0 |                   0.0
| max_leaf_nodes           | int    | 10        |      7.0 |     12.0 |                  11.0
| min_impurity_decrease    | float  | 0.0       |      0.0 |     0.01 | 0.0012521483206650744
| bootstrap                | factor | 1         |      1.0 |      1.0 |                   1.0
| oob_score                | factor | 0         |      1.0 |      1.0 |                   1.0
```

15.10.1 Show variable importance

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

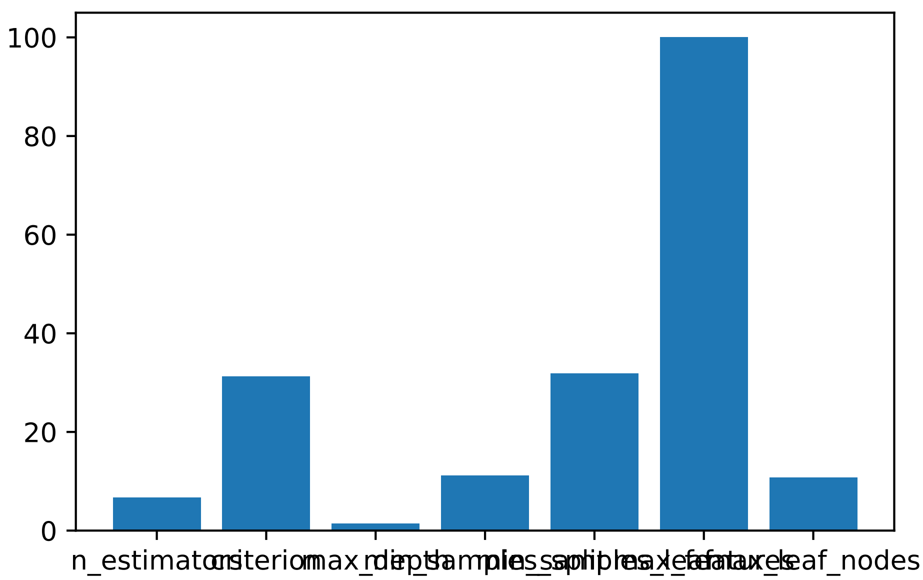


Figure 15.2: Variable importance plot, threshold 0.025.

15.10.2 Get Default Hyperparameters

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

15.10.3 Get SPOT Results

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

```
[[9.00000000e+00 0.00000000e+00 1.10000000e+01 1.90000000e+01
  1.00000000e+00 3.78000081e-03 0.00000000e+00 1.10000000e+01
  1.25214832e-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': 512,
  'criterion': 'gini',
  'max_depth': 2048,
  'min_samples_split': 19,
```

```
 'min_samples_leaf': 1,
 'min_weight_fraction_leaf': 0.0037800008096200988,
 'max_features': 'sqrt',
 'max_leaf_nodes': 2048,
 'min_impurity_decrease': 0.0012521483206650744,
 '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, max_depth=2048, max_leaf_nodes=2048,
                       min_impurity_decrease=0.0012521483206650744,
                       min_samples_split=19,
                       min_weight_fraction_leaf=0.0037800008096200988,
                       n_estimators=512, oob_score=1)
```

15.10.4 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
```

```
0.3493408662900188
```

```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
```

15.10.5 Handling Non-deterministic Results

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

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

```
mean_res: 0.3608913998744507
std_res: 0.008208609206199221
min_res: 0.3483992467043314
max_res: 0.3785310734463277
median_res: 0.358286252354049
```

15.10.6 Evalution of the Default Hyperparameters

```python
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)
```

```
0.32109227871939733
```

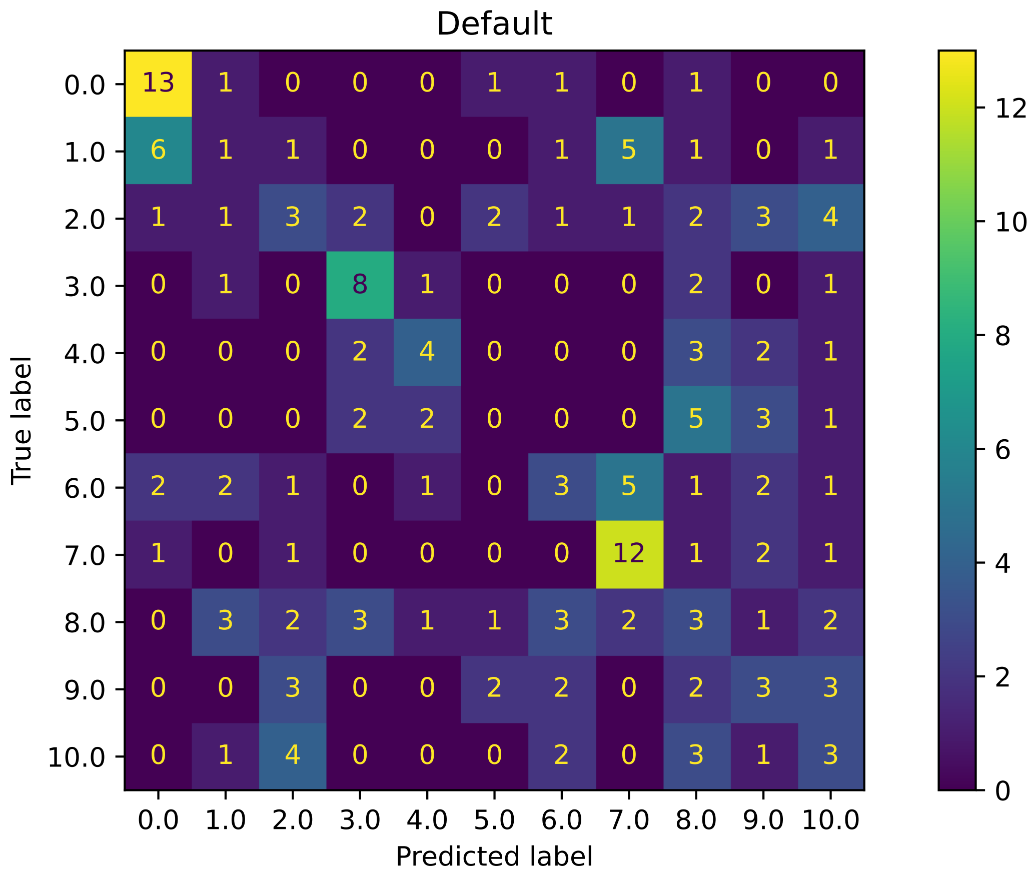
Since one single evaluation is not meaningful, we perform, similar to the evaluation of the SPOT results, $n = 30$ runs of the default setting and and calculate the mean and standard deviation of the performance metric.

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

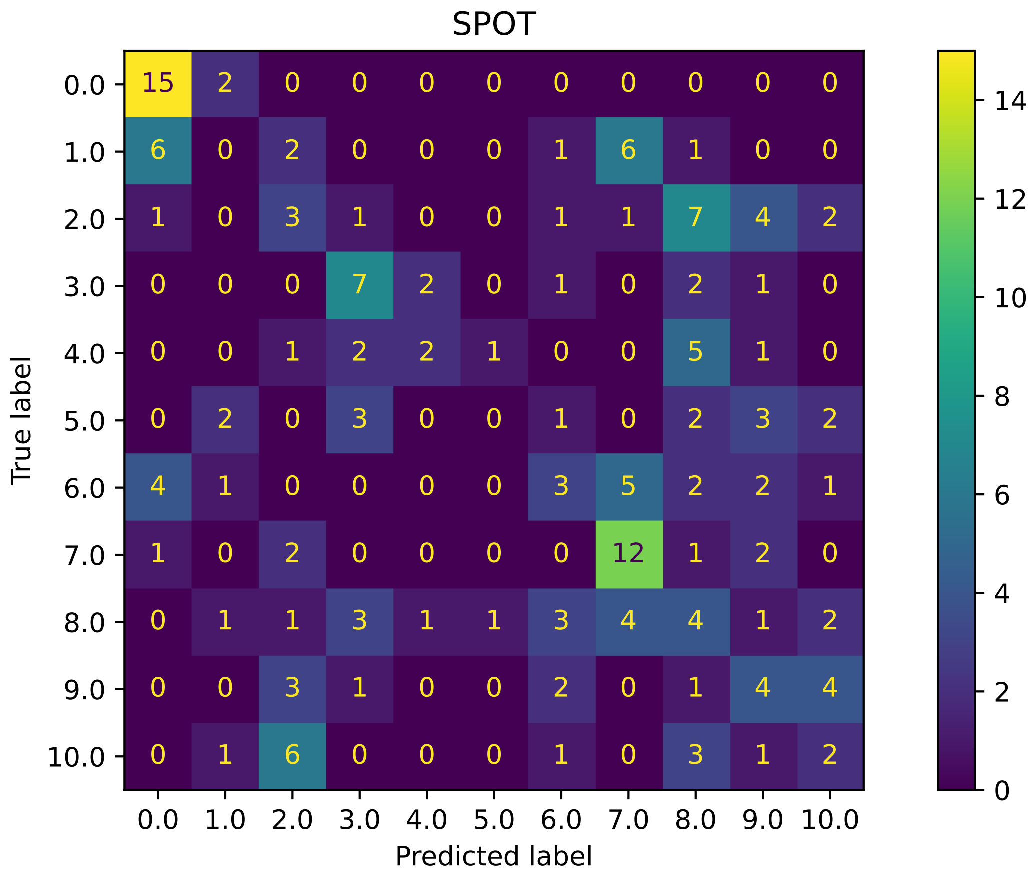
```
mean_res: 0.34425612052730686
std_res: 0.01590120143699079
min_res: 0.3050847457627119
max_res: 0.38323917137476454
median_res: 0.3465160075329567
```

15.10.7 Plot: Compare Predictions

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




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




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

```
(-0.36761006289308173, -0.29968553459119496)
```

15.10.8 Cross-validated Evaluations

```
from spotPython.sklearn.traintest import evaluate_cv
fun_control.update({
     "eval": "train_cv",
     "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)
```

```
(0.369496855345912, None)
```

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

```
(0.30157952069716776, None)
```

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

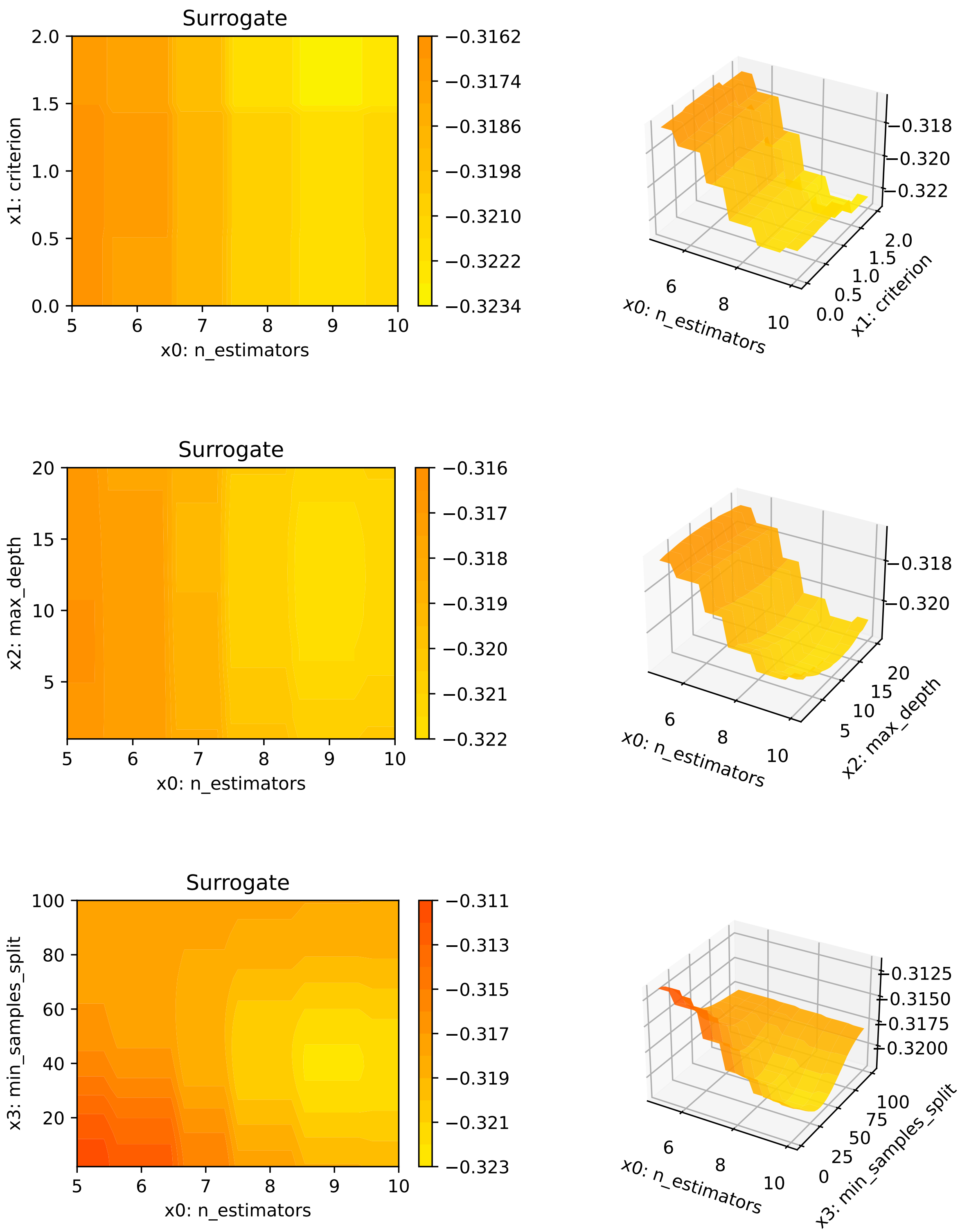
```python
fun_control.update({
     "eval": "data_cv",
     "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)
```

```
(0.3662608987256874, None)
```

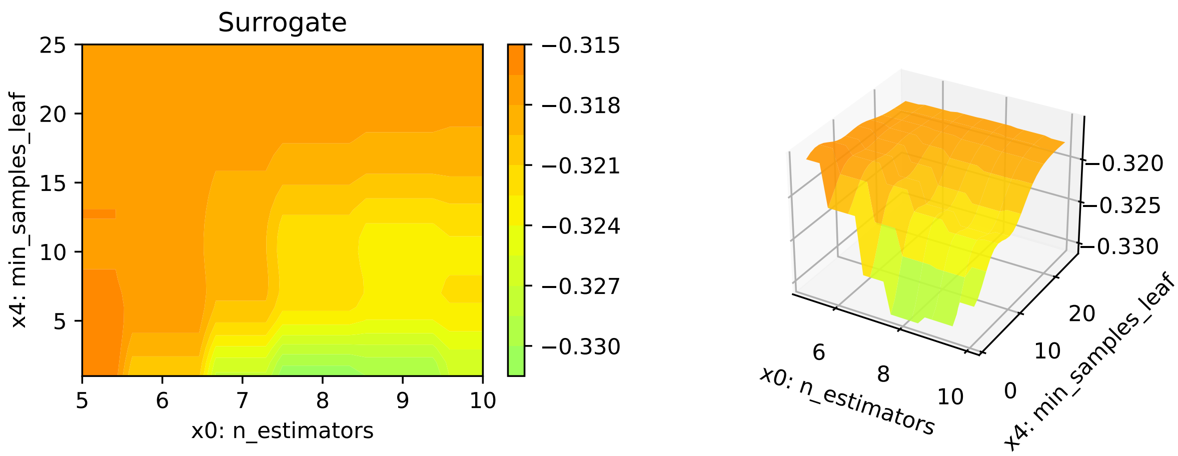
15.10.9 Detailed Hyperparameter Plots

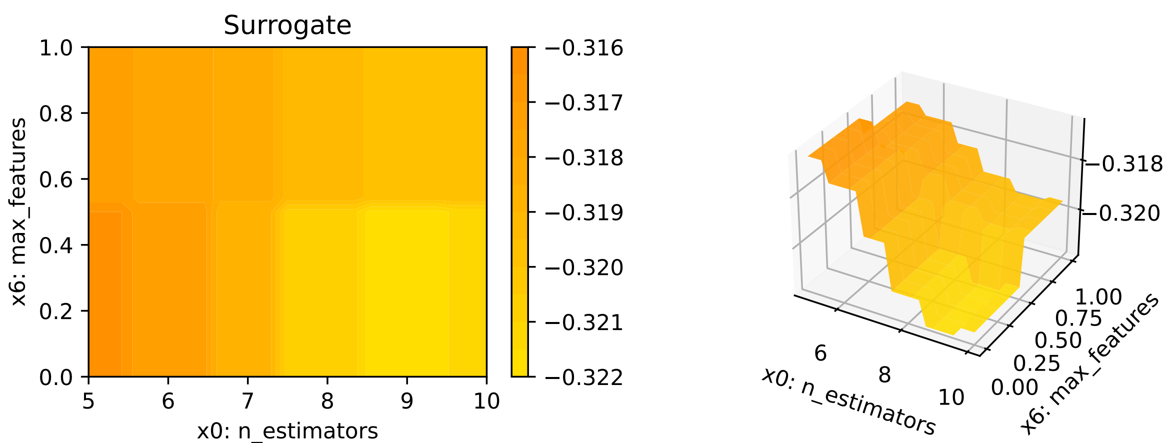
```python
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

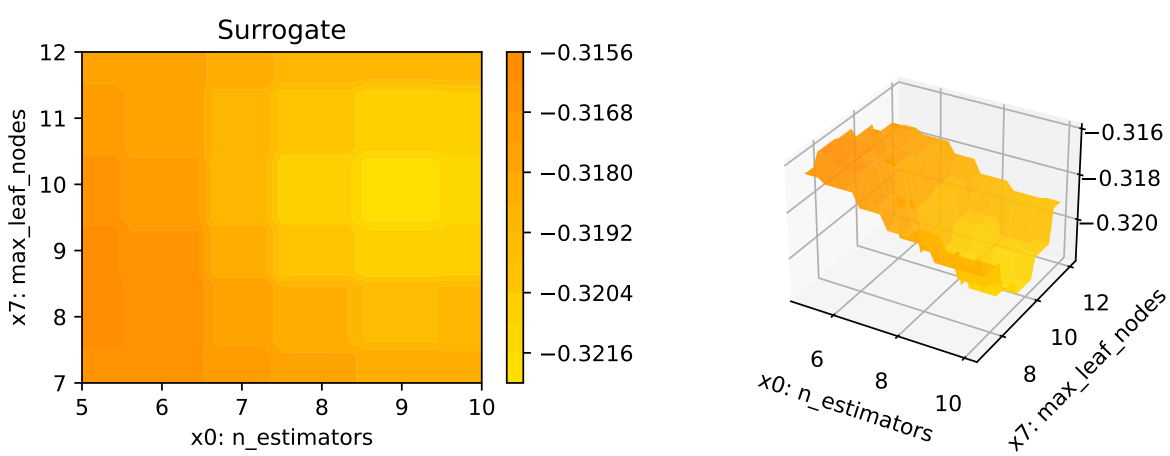
```
n_estimators:  6.713241330765558
criterion:  31.262882253289806
max_depth:  1.4543497171349253
min_samples_split:  11.159874453361999
min_samples_leaf:  31.905517638335457
max_features:  100.0
max_leaf_nodes:  10.721020927384256
```

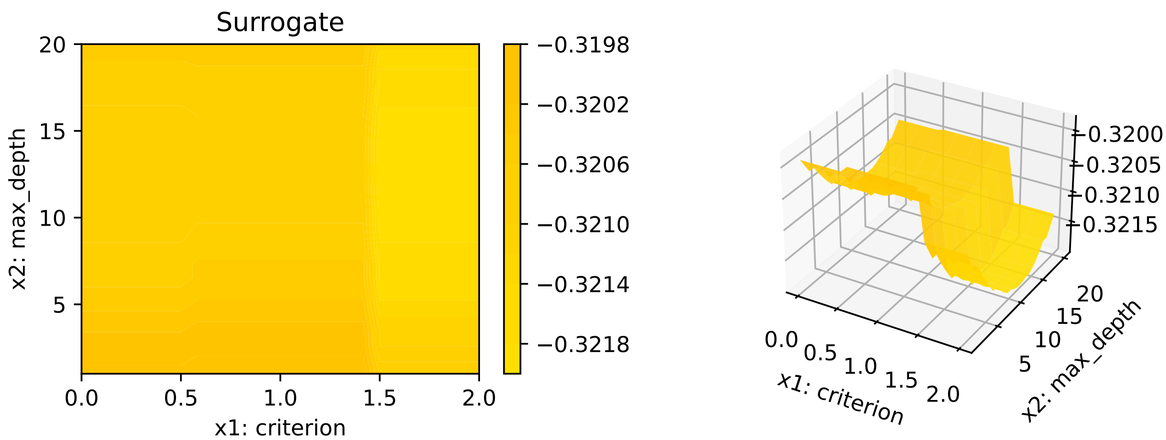


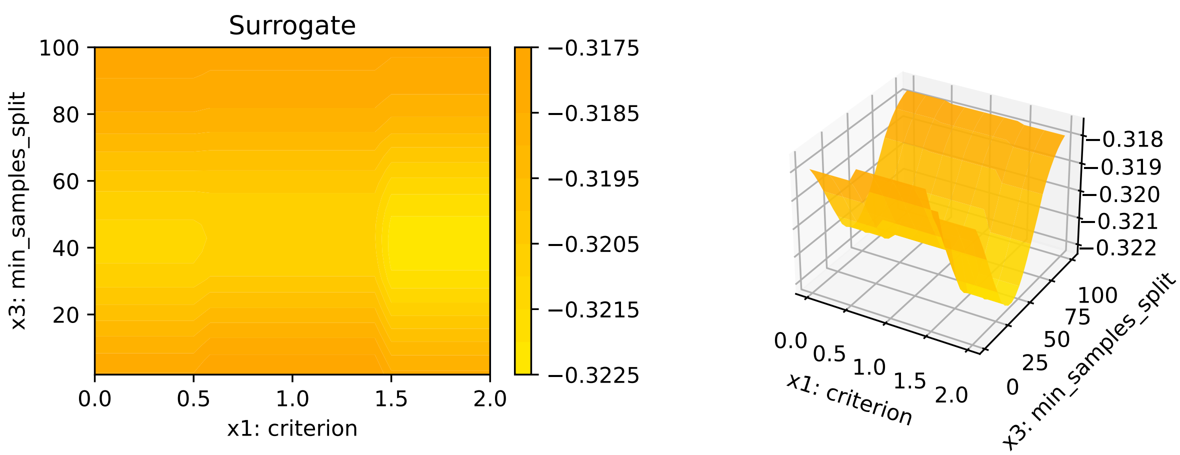
Surrogate
x1: criterion
x0: n_estimators
−0.3162
−0.3174
−0.3186
−0.3198
−0.3210
−0.3222
−0.3234
−0.318
−0.320
−0.322
Surrogate
x2: max_depth
x0: n_estimators
−0.316
−0.317
−0.318
−0.319
−0.320
−0.321
−0.322
Surrogate
x3: min_samples_split
x0: n_estimators
−0.311
−0.313
−0.315
−0.317
−0.319
−0.321
−0.323
−0.3125
−0.3150
−0.3175
−0.3200

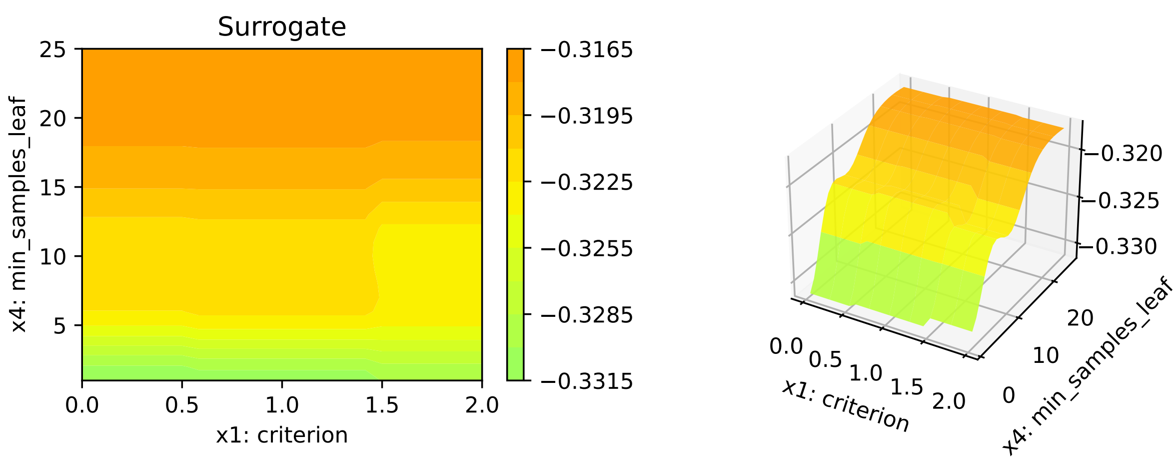

Surrogate
x4: min_samples_leaf
x0: n_estimators
−0.315
−0.318
−0.321
−0.324
−0.327
−0.330
−0.320
−0.325
−0.330
x0: n_estimators
x4: min_samples_leaf

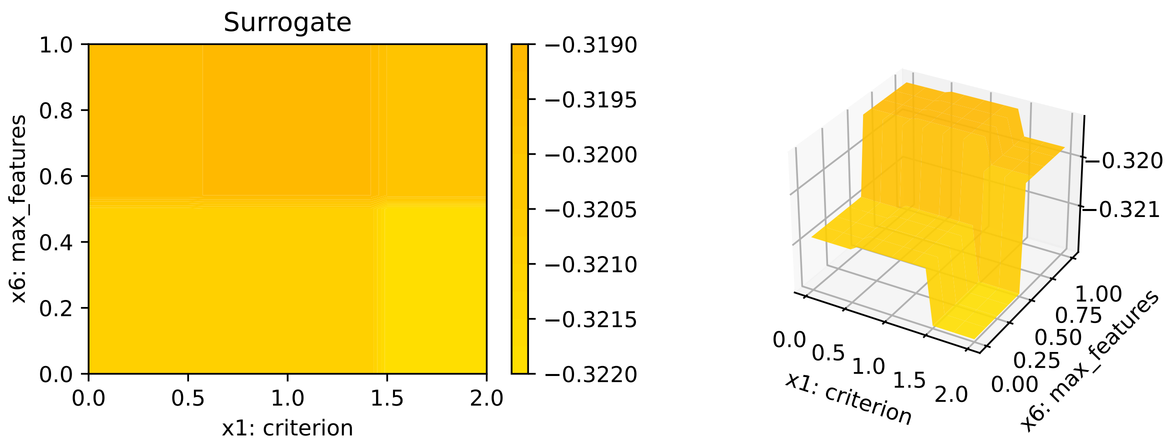

Surrogate
x6: max_features
x0: n_estimators
−0.316
−0.317
−0.318
−0.319
−0.320
−0.321
−0.322
−0.318
−0.320
x0: n_estimators
x6: max_features

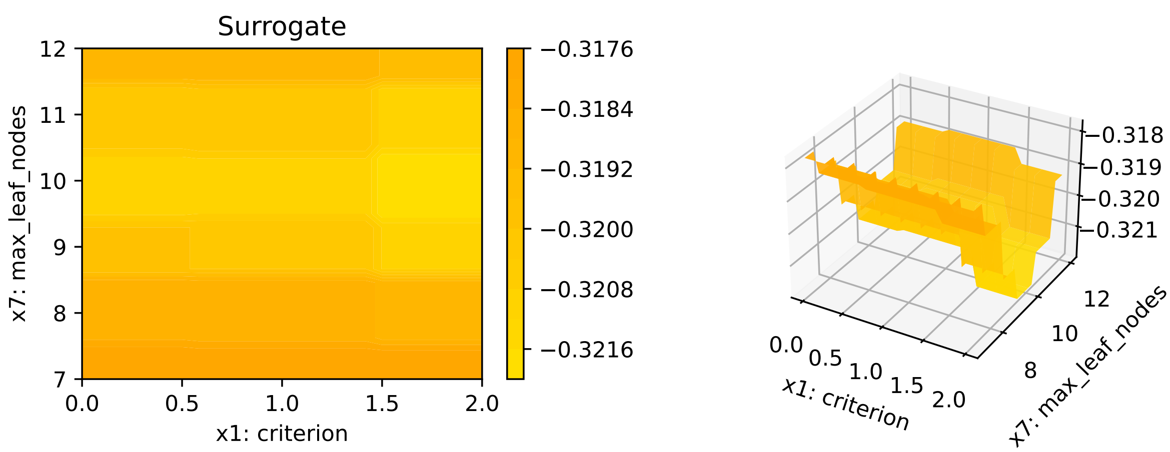

Surrogate
x7: max_leaf_nodes
x0: n_estimators
−0.3156
−0.3168
−0.3180
−0.3192
−0.3204
−0.3216
−0.316
−0.318
−0.320
x0: n_estimators
x7: max_leaf_nodes

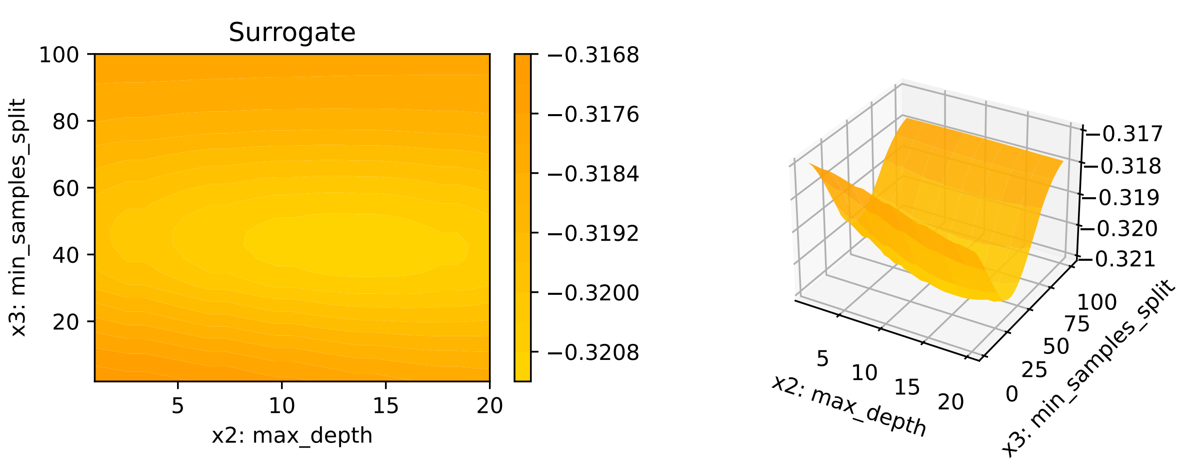

Surrogate
x2: max_depth
x1: criterion
20
15
10
5
0.0
0.5
1.0
1.5
2.0
−0.3198
−0.3202
−0.3206
−0.3210
−0.3214
−0.3218
0.3200
0.3205
0.3210
0.3215

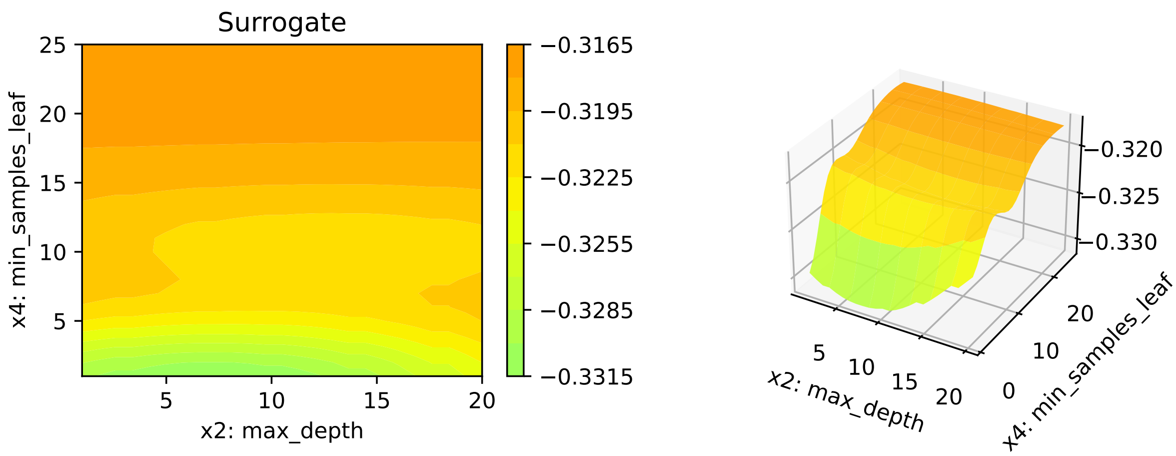

Surrogate
x3: min_samples_split
x1: criterion
100
80
60
40
20
0.0
0.5
1.0
1.5
2.0
−0.3175
−0.3185
−0.3195
−0.3205
−0.3215
−0.3225
−0.318
−0.319
−0.320
−0.321
−0.322
75
50
25
0

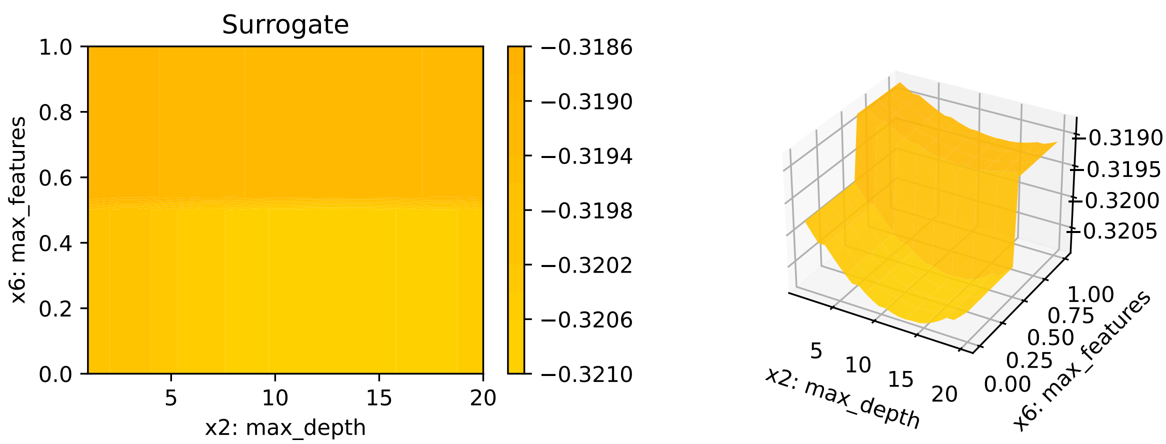

Surrogate
x4: min_samples_leaf
x1: criterion
25
20
15
10
5
0.0
0.5
1.0
1.5
2.0
−0.3165
−0.3195
−0.3225
−0.3255
−0.3285
−0.3315
−0.320
−0.325
−0.330
0

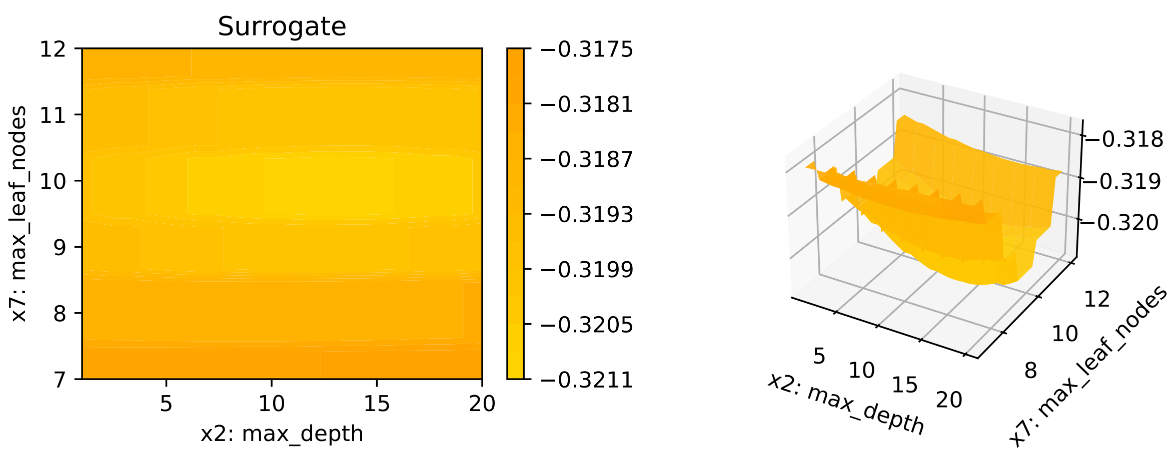

Surrogate
x1: criterion
x6: max_features
0.0
0.5
1.0
1.5
2.0
0.0
0.2
0.4
0.6
0.8
1.0
−0.3190
−0.3195
−0.3200
−0.3205
−0.3210
−0.3215
−0.3220
0.00
0.25
0.50
0.75
1.00
−0.320
−0.321

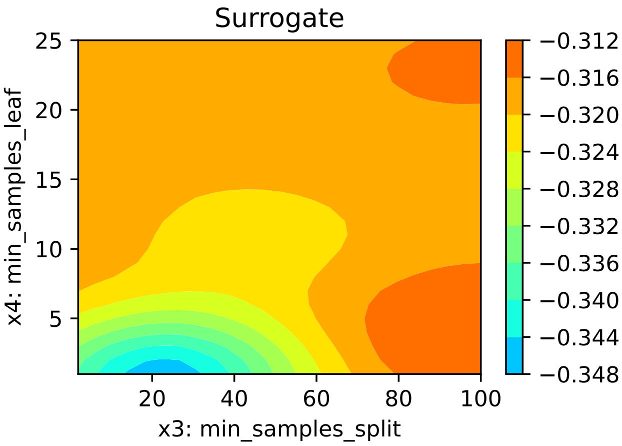

Surrogate
x1: criterion
x7: max_leaf_nodes
0.0
0.5
1.0
1.5
2.0
7
8
9
10
11
12
−0.3176
−0.3184
−0.3192
−0.3200
−0.3208
−0.3216
−0.318
−0.319
−0.320
−0.321

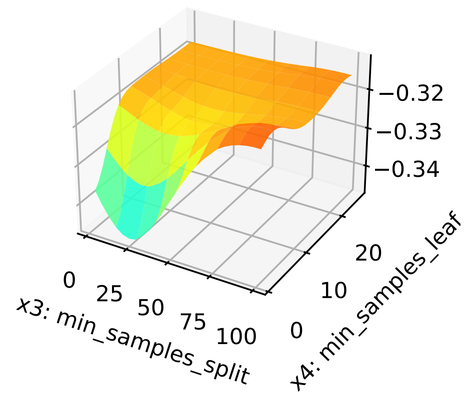

Surrogate
x2: max_depth
x3: min_samples_split
5
10
15
20
25
40
50
60
75
80
100
0
−0.3168
−0.3176
−0.3184
−0.3192
−0.3200
−0.3208
−0.317
−0.318
−0.319
−0.320
−0.321

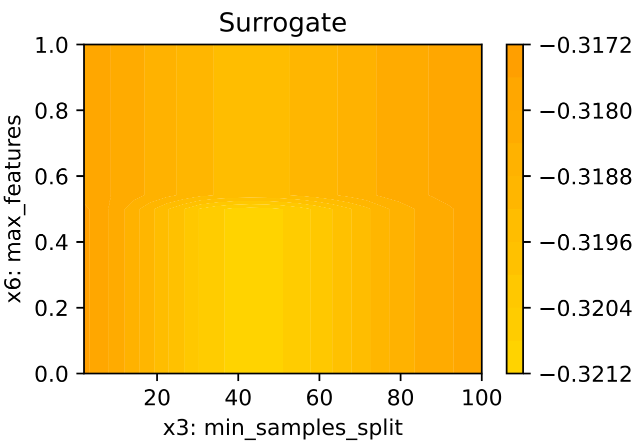

Surrogate
x4: min_samples_leaf
x2: max_depth
−0.3165
−0.3195
−0.3225
−0.3255
−0.3285
−0.3315
−0.320
−0.325
−0.330
x4: min_samples_leaf
x2: max_depth

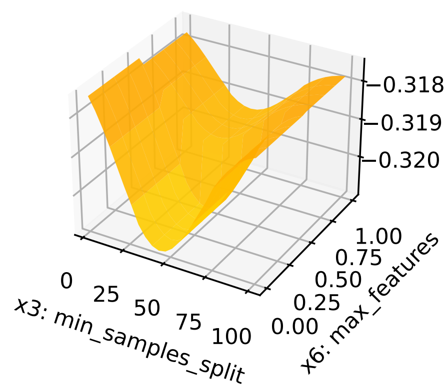

Surrogate
x6: max_features
x2: max_depth
−0.3186
−0.3190
−0.3194
−0.3198
−0.3202
−0.3206
−0.3210
−0.3190
−0.3195
−0.3200
−0.3205
x6: max_features
x2: max_depth

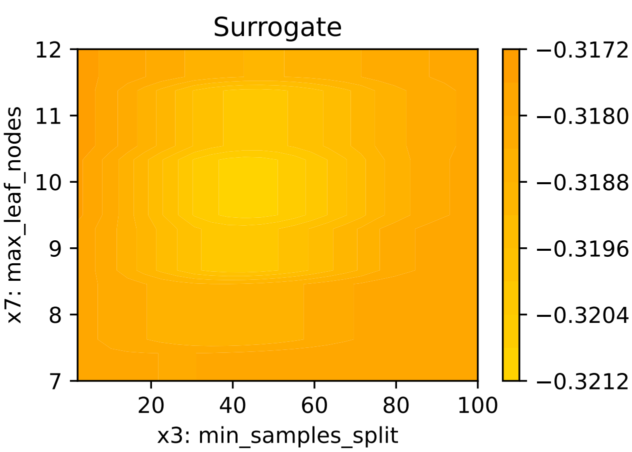

Surrogate
x7: max_leaf_nodes
x2: max_depth
−0.3175
−0.3181
−0.3187
−0.3193
−0.3199
−0.3205
−0.3211
−0.318
−0.319
−0.320
x7: max_leaf_nodes
x2: max_depth

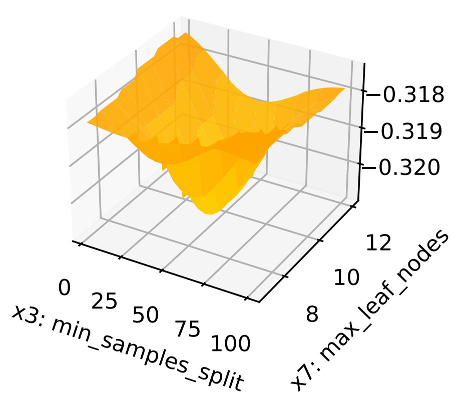

Surrogate
x4: min_samples_leaf
x3: min_samples_split
25
20
15
10
5
20
40
60
80
100
−0.312
−0.316
−0.320
−0.324
−0.328
−0.332
−0.336
−0.340
−0.344
−0.348

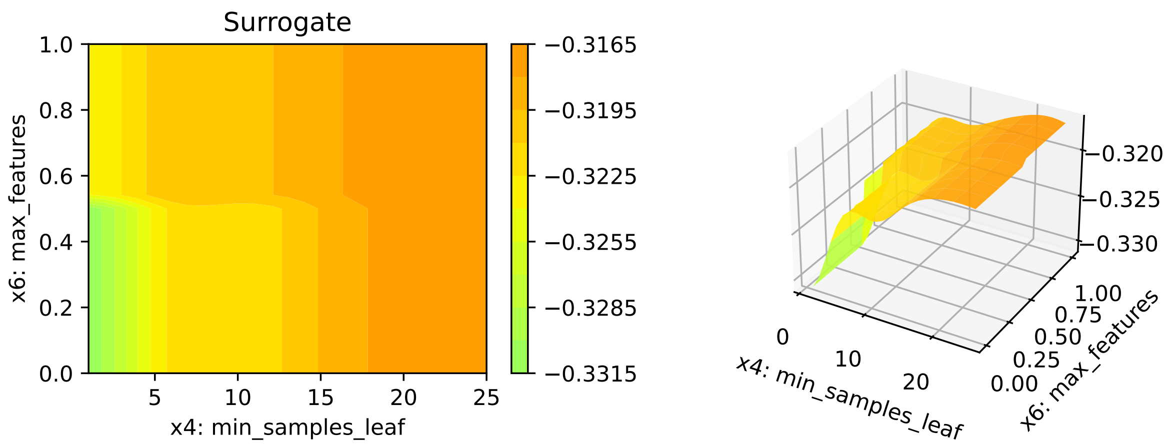

x3: min_samples_split
x4: min_samples_leaf
0
25
50
75
100
0
10
20
−0.32
−0.33
−0.34

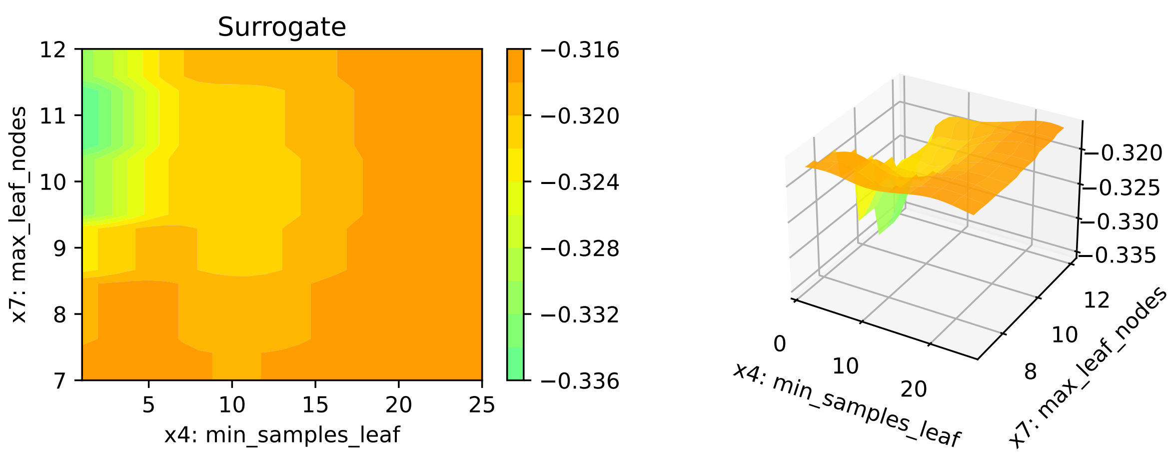

Surrogate
x6: max_features
x3: min_samples_split
1.0
0.8
0.6
0.4
0.2
0.0
20
40
60
80
100
−0.3172
−0.3180
−0.3188
−0.3196
−0.3204
−0.3212

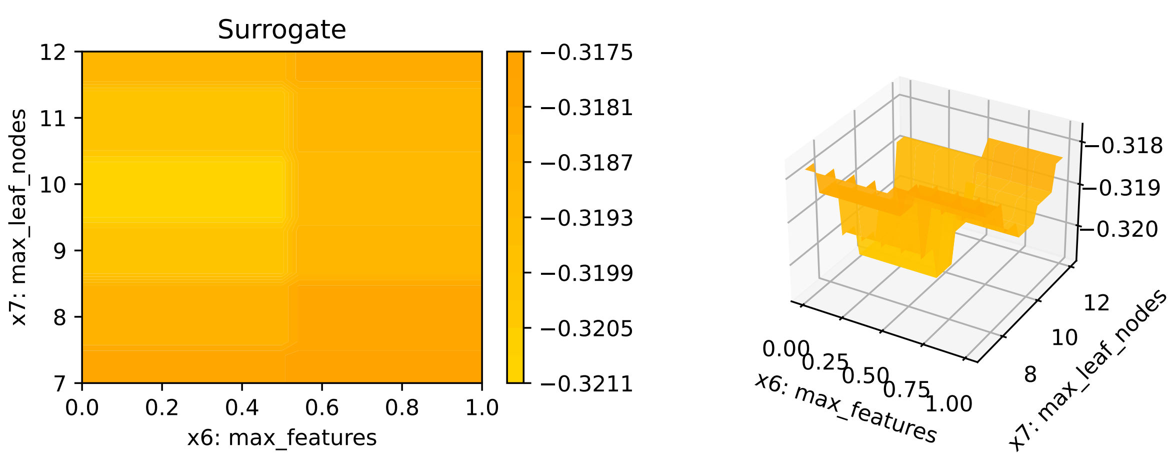

x3: min_samples_split
x6: max_features
0
25
50
75
100
0.00
0.25
0.50
0.75
1.00
−0.318
−0.319
−0.320


Surrogate
x7: max_leaf_nodes
x3: min_samples_split
12
11
10
9
8
7
20
40
60
80
100
−0.3172
−0.3180
−0.3188
−0.3196
−0.3204
−0.3212


x3: min_samples_split
x7: max_leaf_nodes
0
25
50
75
100
8
10
12
−0.318
−0.319
−0.320


Surrogate
x6: max_features
x4: min_samples_leaf
−0.3165
−0.3195
−0.3225
−0.3255
−0.3285
−0.3315
x4: min_samples_leaf
x6: max_features


Surrogate
x7: max_leaf_nodes
x4: min_samples_leaf
−0.316
−0.320
−0.324
−0.328
−0.332
−0.336
x4: min_samples_leaf
x7: max_leaf_nodes


Surrogate
x7: max_leaf_nodes
x6: max_features
−0.3175
−0.3181
−0.3187
−0.3193
−0.3199
−0.3205
−0.3211
x6: max_features
x7: max_leaf_nodes

15.10.10 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

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

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

15.10.11 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

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

16 HPT: sklearn XGB Classifier VBDP Data

This document refers to the following software versions:

- `python`: 3.10.10

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

```
spotPython                0.2.46
spotRiver                 0.0.93
```

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

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

```
!pip install spotPython
```

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

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

16.1 Step 1: Setup

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

```
MAX_TIME = 1
INIT_SIZE = 5
ORIGINAL = False
```

```python
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_2023-06-27_03-57-45
```

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

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

Caution: Tensorboard does not work under Windows

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

```python
from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="classification",
    tensorboard_path="runs/16_spot_hpt_sklearn_classification")
```

16.3 Step 3: PyTorch Data Loading

16.3.1 1. Load Data: Classification VBDP

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

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

```
(707, 65)
```

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	...	x56	x57	x58	x59	x60	x61	x62	x63
0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	...	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0
3	0.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	0.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0

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

16.3.2 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

```python
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

```python
# add the dataset to the fun_control
fun_control.update({"data": train_df, # full dataset,
               "train": train,
               "test": test,
               "n_samples": n_samples,
               "target_column": target_column})
```

16.4 Step 4: Specification of the Preprocessing Model

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

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

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

```
# prep_model = StandardScaler()
# fun_control.update({"prep_model": prep_model})
```

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

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

16.5 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
```

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

16.6.1 Modify hyperparameter of type numeric and integer (boolean)

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

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

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

16.6.2 Modify hyperparameter of type factor

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

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

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

The new setting can be controlled via:

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

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

16.6.3 Optimizers

Optimizers are described in Section 14.6.1.

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

16.7.1 Evaluation

The evaluation procedure requires the specification of two elements:

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

16.7.2 Selection of the Objective: Metric and Loss Functions

- Machine learning models are optimized with respect to a metric, for example, the `accuracy` function.
- Deep learning, e.g., neural networks are optimized with respect to a loss function, for example, the `cross_entropy` function and evaluated with respect to a metric, for example, the `accuracy` function.

16.7.3 Loss Function

The loss function, that is usually used in deep learning for optimizing the weights of the net, is stored in the `fun_control` dictionary as `"loss_function"`.

16.7.4 Metric Function

There are two different types of metrics in `spotPython`:

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

We will consider multi-class classification metrics, e.g., `mapk_score` and `top_k_accuracy_score`.

Predict Probabilities

In this multi-class classification example the machine learning algorithm should return the probabilities of the specific classes (`"predict_proba"`) instead of the predicted values.

We set `"predict_proba"` to `True` in the `fun_control` dictionary.

16.7.4.1 The MAPK Metric

To select the MAPK metric, the following two entries can be added to the `fun_control` dictionary:

`"metric_sklearn": mapk_score"`

`"metric_params": {"k": 3}`.

16.7.4.2 Other Metrics

Alternatively, other metrics for multi-class classification can be used, e.g.,: * top_k_accuracy_score or * roc_auc_score

The metric `roc_auc_score` requires the parameter `"multi_class"`, e.g.,

`"multi_class": "ovr"`.

This is set in the `fun_control` dictionary.

Weights

`spotPython` performs a minimization, therefore, metrics that should be maximized have to be multiplied by -1. This is done by setting `"weights"` to `-1`.

- The complete setup for the metric in our example is:

```
from spotPython.utils.metrics import mapk_score
fun_control.update({
               "weights": -1,
               "metric_sklearn": mapk_score,
               "predict_proba": True,
               "metric_params": {"k": 3},
               })
```

16.7.5 Evaluation on Hold-out Data

- The default method for computing the performance is `"eval_holdout"`.
- Alternatively, cross-validation can be used for every machine learning model.
- Specifically for RandomForests, the OOB-score can be used.

```
fun_control.update({
    "eval": "train_hold_out",
})
```

16.7.5.1 Cross Validation

Instead of using the OOB-score, the classical cross validation can be used. The number of folds is set by the key `"k_folds"`. For example, to use 5-fold cross validation, the key `"k_folds"` is set to `5`. Uncomment the following line to use cross validation:

```
# fun_control.update({
#      "eval": "train_cv",
#      "k_folds": 10,
# })
```

16.8 Step 8: Calling the SPOT Function

16.8.1 Preparing the SPOT Call

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

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

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

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

16.8.2 The Objective Function

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

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

16.8.3 Run the Spot Optimizer

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

```python
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: -0.36842105263157887 [----------] 1.96%

spotPython tuning: -0.36842105263157887 [#---------] 6.60%

spotPython tuning: -0.36842105263157887 [##--------] 16.52%

spotPython tuning: -0.36842105263157887 [##--------] 18.98%

spotPython tuning: -0.36842105263157887 [###-------] 31.19%
```

```
spotPython tuning: -0.36842105263157887 [###-------] 32.90%

spotPython tuning: -0.36842105263157887 [####------] 36.52%

spotPython tuning: -0.36842105263157887 [####------] 39.66%

spotPython tuning: -0.36842105263157887 [####------] 43.83%

spotPython tuning: -0.36842105263157887 [#####-----] 47.88%

spotPython tuning: -0.36842105263157887 [#####-----] 52.07%

spotPython tuning: -0.36842105263157887 [######----] 56.04%

spotPython tuning: -0.36842105263157887 [######----] 62.55%

spotPython tuning: -0.36842105263157887 [#######---] 70.75%

spotPython tuning: -0.36842105263157887 [########--] 78.83%

spotPython tuning: -0.36842105263157887 [########--] 83.88%

spotPython tuning: -0.3684210526315789 [#########-] 89.86%

spotPython tuning: -0.3684210526315789 [##########] 96.86%

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

<spotPython.spot.spot.Spot at 0x16b1c7880>
```

16.9 Step 9: Tensorboard

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

16.10 Step 10: Results

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

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

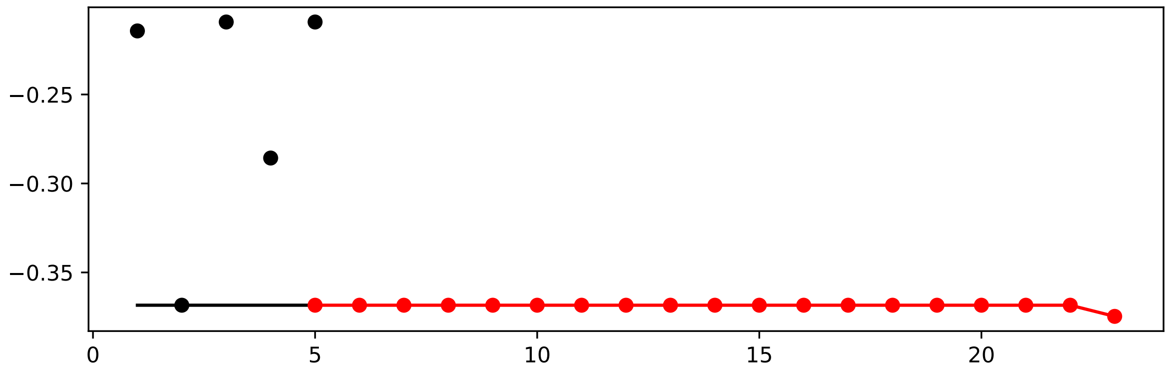


Figure 16.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- Print the results

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

```
| name                | type   | default  |   lower |   upper |                  tuned | trans
|---------------------|--------|----------|---------|---------|------------------------|------
| loss                | factor | log_loss |     0.0 |     0.0 |                    0.0 | None
| learning_rate       | float  | -1.0     |    -5.0 |     0.0 |    -0.5750835072095268 | trans
| max_iter            | int    | 7        |     3.0 |    10.0 |                    8.0 | trans
| max_leaf_nodes      | int    | 5        |     1.0 |    12.0 |                    4.0 | trans
| max_depth           | int    | 2        |     1.0 |    20.0 |                   20.0 | trans
| min_samples_leaf    | int    | 4        |     2.0 |    10.0 |                    2.0 | trans
| l2_regularization   | float  | 0.0      |     0.0 |    10.0 |     3.5762246122128127 | None
| max_bins            | int    | 255      |   127.0 |   255.0 |                  131.0 | None
| early_stopping      | factor | 1        |     0.0 |     1.0 |                    1.0 | None
| n_iter_no_change    | int    | 10       |     5.0 |    20.0 |                    5.0 | None
| tol                 | float  | 0.0001   |   1e-05 |   0.001 | 8.099608041927449e-05  | None
```

16.10.1 Show variable importance

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

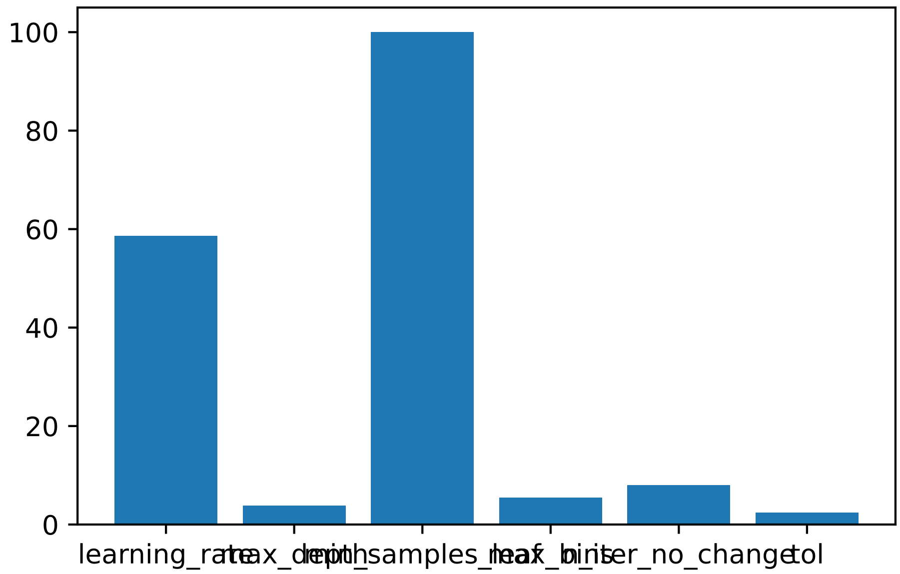


Figure 16.2: Variable importance plot, threshold 0.025.

16.10.2 Get Default Hyperparameters

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

16.10.3 Get SPOT Results

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

```
[[ 0.00000000e+00 -5.75083507e-01  8.00000000e+00  4.00000000e+00
   2.00000000e+01  2.00000000e+00  3.57622461e+00  1.31000000e+02
   1.00000000e+00  5.00000000e+00  8.09960804e-05]]
```

```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': 0.2660213498233799,
  'max_iter': 256,
  'max_leaf_nodes': 16,
  'max_depth': 1048576,
  'min_samples_leaf': 4,
  'l2_regularization': 3.5762246122128127,
  'max_bins': 131,
  'early_stopping': 1,
  'n_iter_no_change': 5,
  'tol': 8.099608041927449e-05}]
```
```

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

```
HistGradientBoostingClassifier(early_stopping=1,
                               l2_regularization=3.5762246122128127,
                               learning_rate=0.2660213498233799, max_bins=131,
                               max_depth=1048576, max_iter=256,
                               max_leaf_nodes=16, min_samples_leaf=4,
                               n_iter_no_change=5, tol=8.099608041927449e-05)
```

16.10.4 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
```

```
0.3436911487758945
```

```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
```

16.10.5 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: 0.34199623352165726
std_res: 0.015627103060093243
min_res: 0.3163841807909605
max_res: 0.371939736346516
median_res: 0.3436911487758946
```

16.10.6 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)
```

```
0.3163841807909605
```

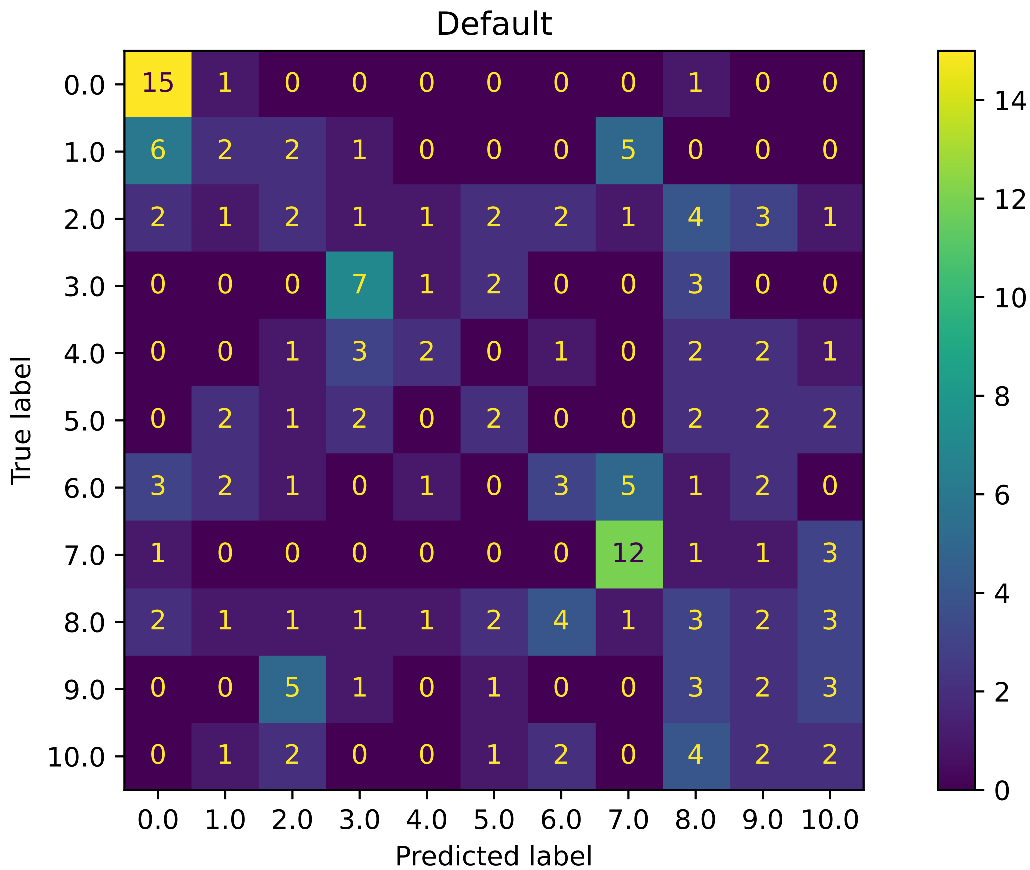
Since one single evaluation is not meaningful, we perform, similar to the evaluation of the SPOT results, $n = 30$ runs of the default setting and and calculate the mean and standard deviation of the performance metric.

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

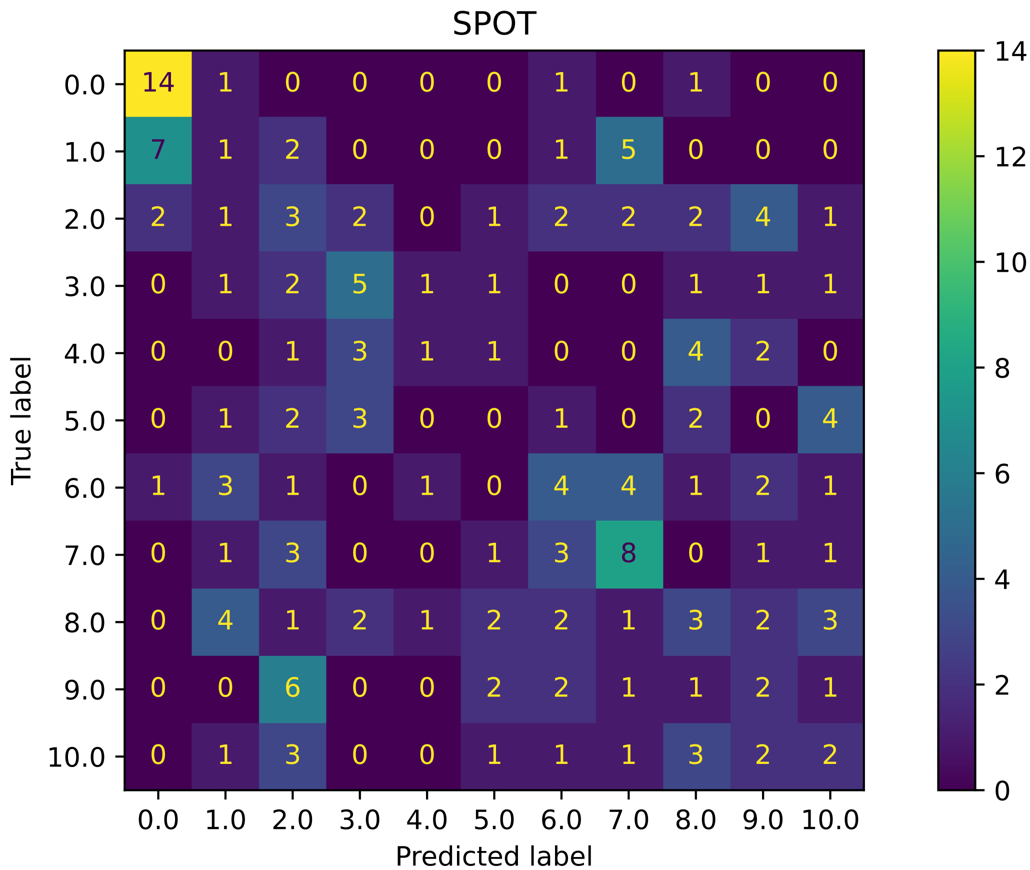
```
mean_res: 0.34375392341494043
std_res: 0.01634179517441997
min_res: 0.3116760828625236
max_res: 0.3775894538606403
median_res: 0.3451035781544256
```

16.10.7 Plot: Compare Predictions

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




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




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

```
(-0.3746867167919799, -0.20927318295739344)
```

16.10.8 Cross-validated Evaluations

```
from spotPython.sklearn.traintest import evaluate_cv
fun_control.update({
     "eval": "train_cv",
     "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)
```

```
(0.3377358490566038, None)
```

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

```
(0.2723856209150327, None)
```

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

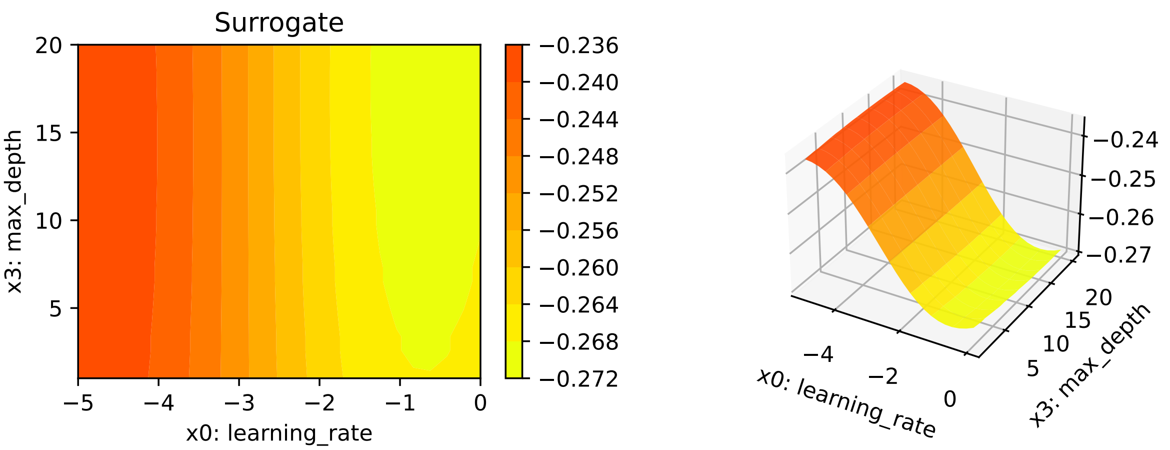
```
fun_control.update({
     "eval": "data_cv",
     "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)
```

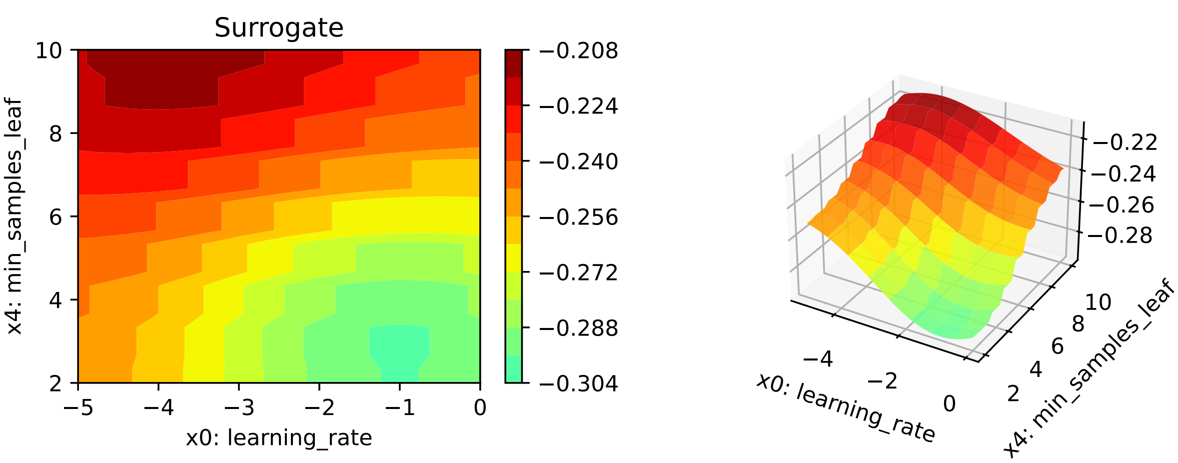
```
(0.33852112676056334, None)
```

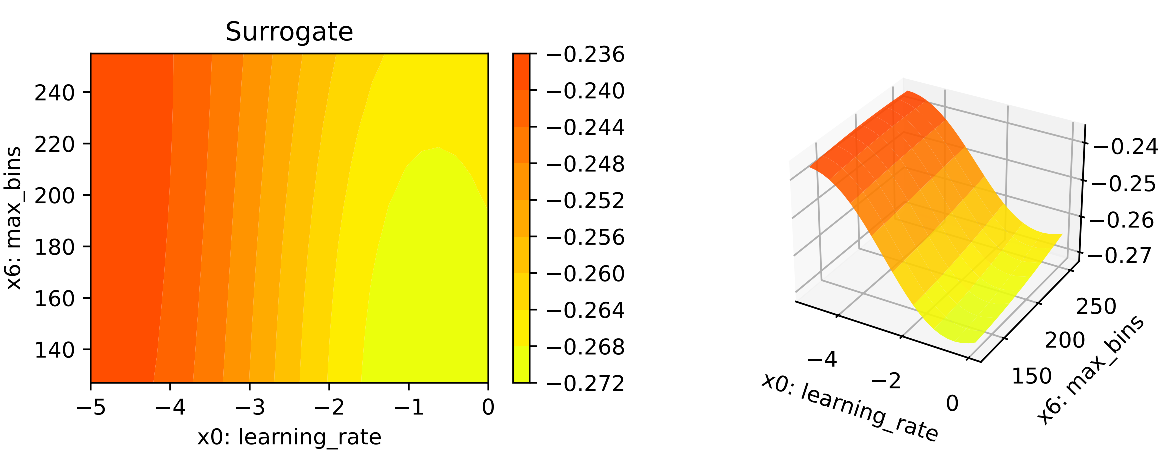
16.10.9 Detailed Hyperparameter Plots

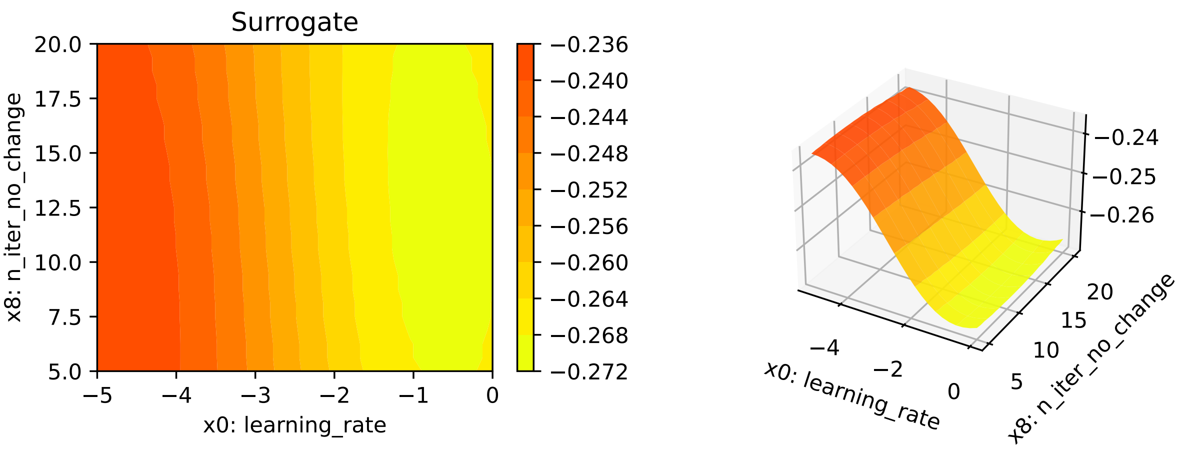
```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

```
learning_rate:  58.57413378906925
max_depth:  3.8503560508202916
min_samples_leaf:  100.0
max_bins:  5.477500661012056
n_iter_no_change:  7.929455352209658
tol:  2.3680252056886846
```

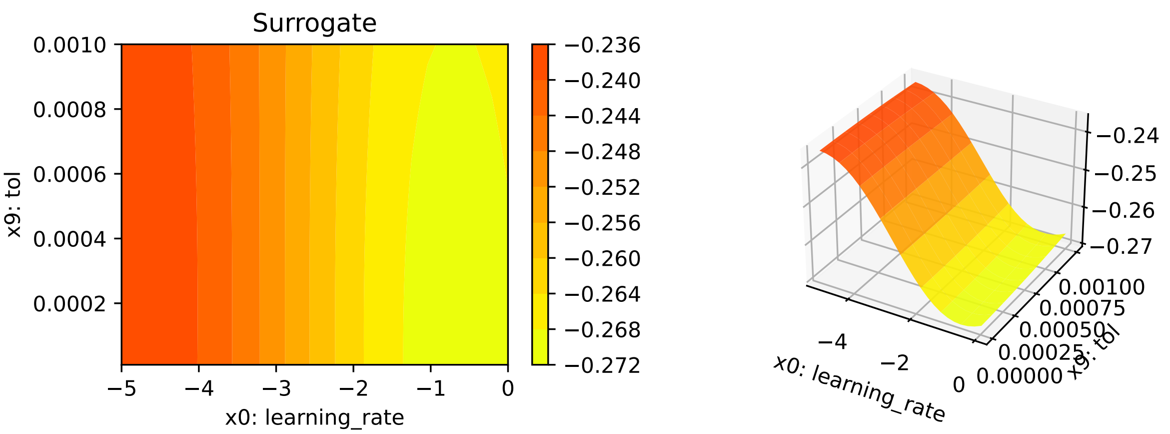

Surrogate
x3: max_depth
x0: learning_rate
20
15
10
5
−5
−4
−3
−2
−1
0
−0.236
−0.240
−0.244
−0.248
−0.252
−0.256
−0.260
−0.264
−0.268
−0.272
−0.24
−0.25
−0.26
−0.27
−4
−2
0
x0: learning_rate
5
10
15
20
x3: max_depth


Surrogate
x4: min_samples_leaf
x0: learning_rate
10
8
6
4
2
−5
−4
−3
−2
−1
0
−0.208
−0.224
−0.240
−0.256
−0.272
−0.288
−0.304
−0.22
−0.24
−0.26
−0.28
−4
−2
0
x0: learning_rate
2
4
6
8
10
x4: min_samples_leaf

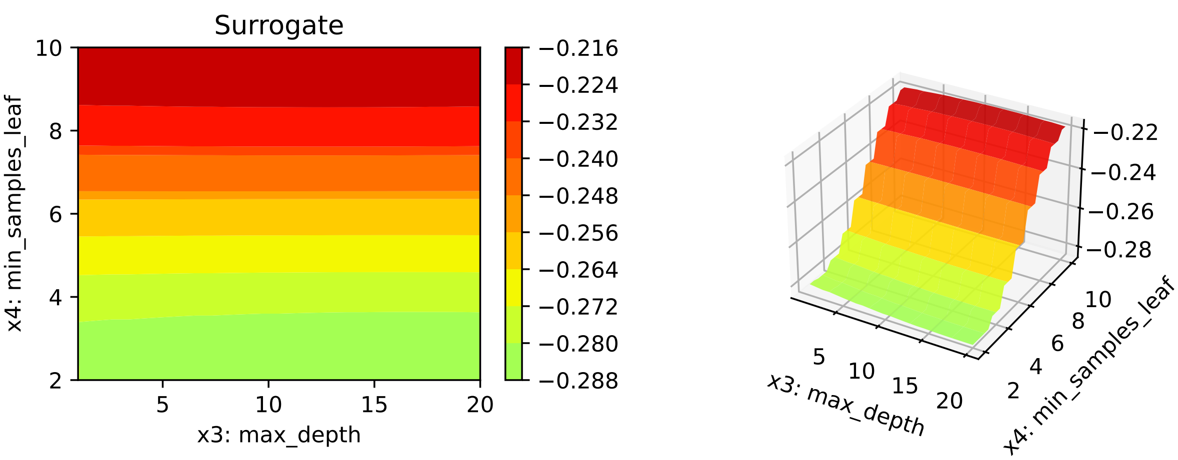

Surrogate
x6: max_bins
x0: learning_rate
240
220
200
180
160
140
−5
−4
−3
−2
−1
0
−0.236
−0.240
−0.244
−0.248
−0.252
−0.256
−0.260
−0.264
−0.268
−0.272
−0.24
−0.25
−0.26
−0.27
−4
−2
0
x0: learning_rate
150
200
250
x6: max_bins



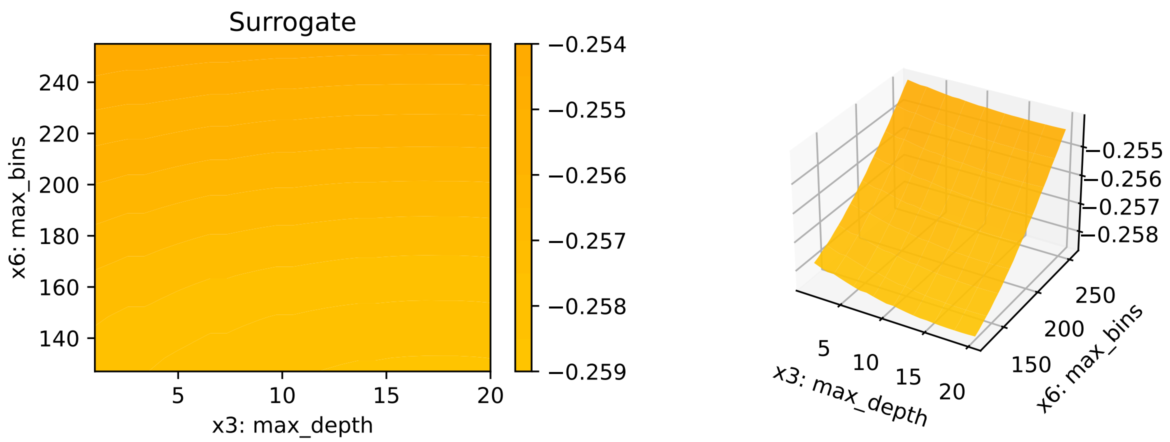
Surrogate
x8: n_iter_no_change
x0: learning_rate
20.0
17.5
15.0
12.5
10.0
7.5
5.0
−5
−4
−3
−2
−1
0
−0.236
−0.240
−0.244
−0.248
−0.252
−0.256
−0.260
−0.264
−0.268
−0.272
−0.24
−0.25
−0.26
20
15
10
5

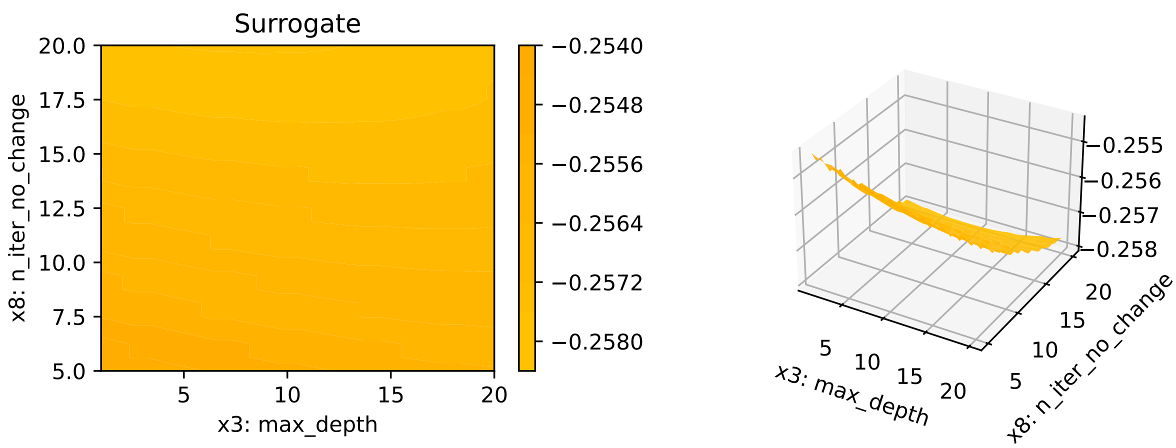


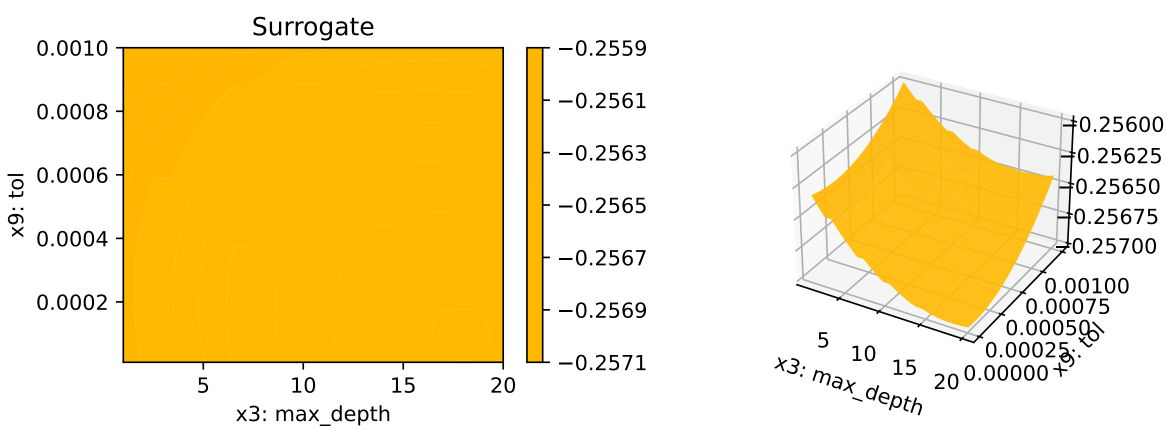
Surrogate
x9: tol
x0: learning_rate
0.0010
0.0008
0.0006
0.0004
0.0002
−5
−4
−3
−2
−1
0
−0.236
−0.240
−0.244
−0.248
−0.252
−0.256
−0.260
−0.264
−0.268
−0.272
−0.24
−0.25
−0.26
−0.27
0.00100
0.00075
0.00050
0.00025
0.00000

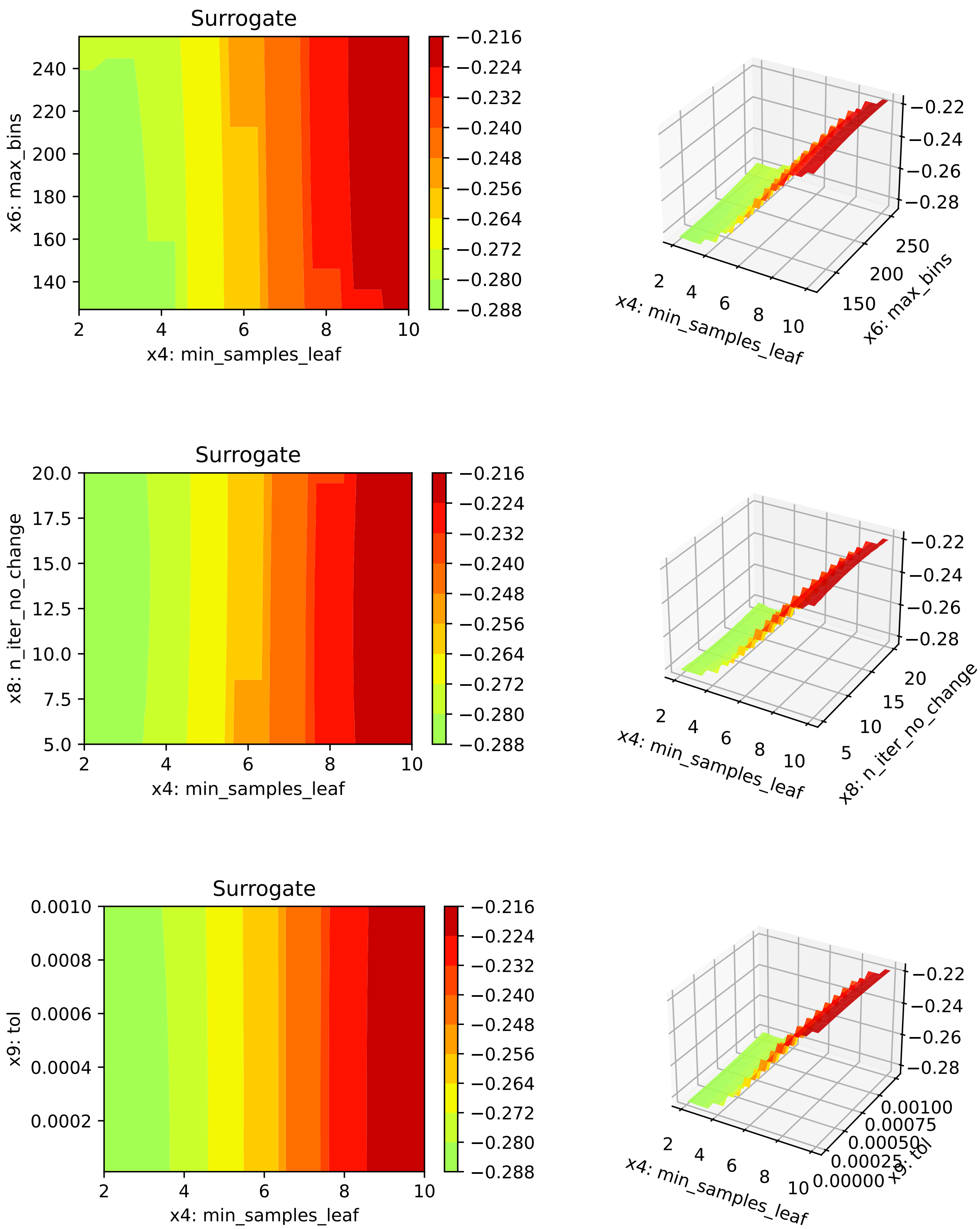


Surrogate
x4: min_samples_leaf
x3: max_depth
10
8
6
4
2
5
10
15
20
−0.216
−0.224
−0.232
−0.240
−0.248
−0.256
−0.264
−0.272
−0.280
−0.288
−0.22
−0.24
−0.26
−0.28

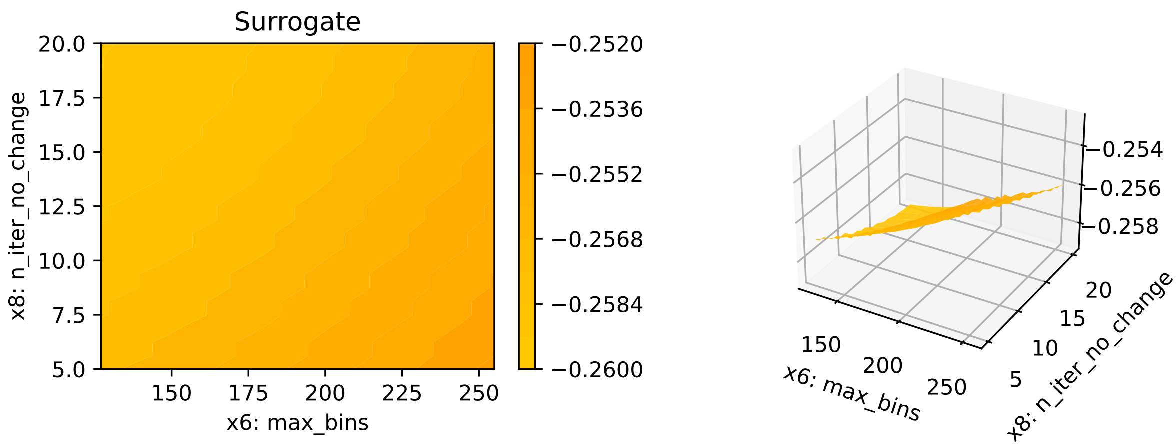

Surrogate
x6: max_bins
x3: max_depth
240
220
200
180
160
140
5
10
15
20
−0.254
−0.255
−0.256
−0.257
−0.258
−0.259
250
200
150

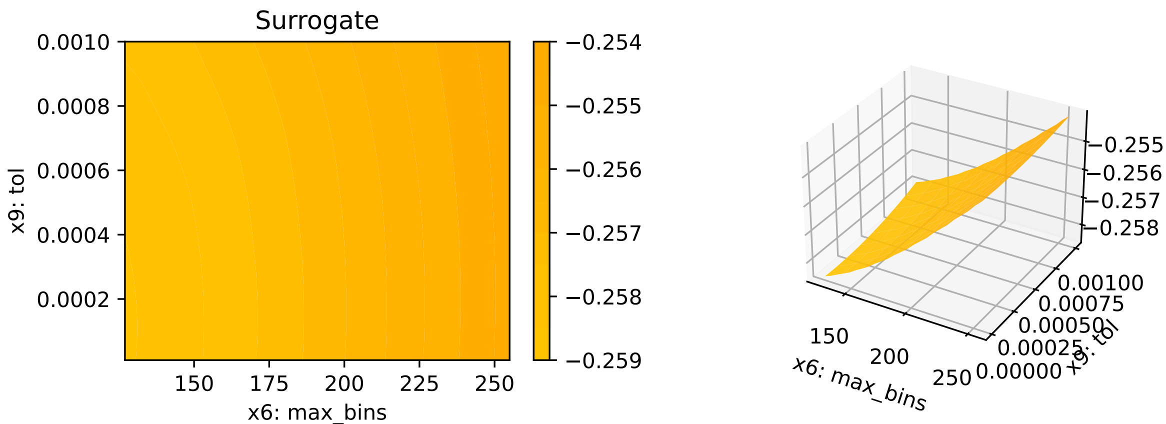

Surrogate
x8: n_iter_no_change
x3: max_depth
20.0
17.5
15.0
12.5
10.0
7.5
5.0
−0.2540
−0.2548
−0.2556
−0.2564
−0.2572
−0.2580
−0.255
−0.256
−0.257
−0.258

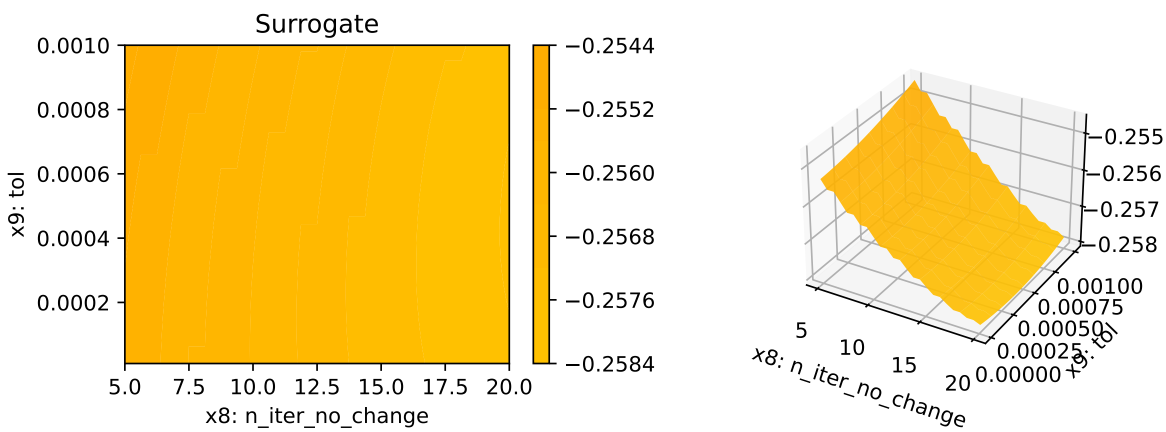

Surrogate
x9: tol
x3: max_depth
0.0010
0.0008
0.0006
0.0004
0.0002
−0.2559
−0.2561
−0.2563
−0.2565
−0.2567
−0.2569
−0.2571
0.25600
0.25625
0.25650
0.25675
0.25700
0.00100
0.00075
0.00050
0.00025
0.00000



Surrogate
x6: max_bins
240
220
200
180
160
140
2
4
6
8
10
x4: min_samples_leaf
−0.216
−0.224
−0.232
−0.240
−0.248
−0.256
−0.264
−0.272
−0.280
−0.288
−0.22
−0.24
−0.26
−0.28
250
200
150
x6: max_bins
Surrogate
x8: n_iter_no_change
20.0
17.5
15.0
12.5
10.0
7.5
5.0
20
15
10
5
x8: n_iter_no_change
Surrogate
x9: tol
0.0010
0.0008
0.0006
0.0004
0.0002
0.00100
0.00075
0.00050
0.00025
0.00000


Surrogate
x6: max_bins
x8: n_iter_no_change
−0.2520
−0.2536
−0.2552
−0.2568
−0.2584
−0.2600
−0.254
−0.256
−0.258


Surrogate
x6: max_bins
x9: tol
−0.254
−0.255
−0.256
−0.257
−0.258
−0.259


Surrogate
x8: n_iter_no_change
x9: tol
−0.2544
−0.2552
−0.2560
−0.2568
−0.2576
−0.2584
−0.255
−0.256
−0.257
−0.258

16.10.10 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

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

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

16.10.11 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

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

17 HPT: sklearn SVC VBDP Data

This document refers to the following software versions:

- `python`: 3.10.10

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

```
spotPython                    0.2.46

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 and the initial design size.

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

```python
ORIGINAL = False
```

```python
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_2023-06-27_04-01-54
```

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

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

Caution: Tensorboard does not work under Windows

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

```python
from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="classification",
    tensorboard_path="runs/16_spot_hpt_sklearn_classification")
```

17.3 Step 3: PyTorch Data Loading

17.3.1 1. Load Data: Classification VBDP

```
import pandas as pd
if ORIGINAL == True:
    train_df = pd.read_csv('./data/VBDP/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`.

17.3.2 Holdout Train and Test Data

We split out a hold-out test set (25% of the data) so we can calculate an example MAP@K

```
import numpy as np
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(train_df.drop(target_column, axis=1),
                                                    random_state=42,
                                                    test_size=0.25,
                                                    stratify=train_df[target_column])
train = pd.DataFrame(np.hstack((X_train, np.array(y_train).reshape(-1, 1))))
test = pd.DataFrame(np.hstack((X_test, np.array(y_test).reshape(-1, 1))))
train.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
test.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
print(train.shape)
print(test.shape)
train.head()
```

```
(530, 65)
(177, 65)
```

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	...	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})
```

17.4 Step 4: Specification of the Preprocessing Model

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

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

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

```
# prep_model = StandardScaler()
# fun_control.update({"prep_model": prep_model})
```

Even more complicated pre-processing steps are possible, e.g., the 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(),
#     )
```

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

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

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

Other core_models are, e.g.,:

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

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

```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

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

17.6.1 Modify hyperparameter of type numeric and integer (boolean)

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

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

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

17.6.2 Modify hyperparameter of type factor

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

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

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

The new setting can be controlled via:

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

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

17.6.3 Optimizers

Optimizers are described in Section 14.6.1.

17.6.4 Selection of the Objective: Metric and Loss Functions

- Machine learning models are optimized with respect to a metric, for example, the `accuracy` function.
- Deep learning, e.g., neural networks are optimized with respect to a loss function, for example, the `cross_entropy` function and evaluated with respect to a metric, for example, the `accuracy` function.

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

The loss function, that is usually used in deep learning for optimizing the weights of the net, is stored in the `fun_control` dictionary as `"loss_function"`.

17.7.1 Metric Function

There are two different types of metrics in `spotPython`:

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

We will consider multi-class classification metrics, e.g., `mapk_score` and `top_k_accuracy_score`.

> **Predict Probabilities**
>
> In this multi-class classification example the machine learning algorithm should return the probabilities of the specific classes (`"predict_proba"`) instead of the predicted values.

We set `"predict_proba"` to `True` in the `fun_control` dictionary.

17.7.1.1 The MAPK Metric

To select the MAPK metric, the following two entries can be added to the `fun_control` dictionary:

`"metric_sklearn": mapk_score"`

`"metric_params": {"k": 3}`.

17.7.1.2 Other Metrics

Alternatively, other metrics for multi-class classification can be used, e.g.,: * top_k_accuracy_score or * roc_auc_score

The metric `roc_auc_score` requires the parameter `"multi_class"`, e.g.,

`"multi_class": "ovr"`.

This is set in the `fun_control` dictionary.

> **Weights**
>
> `spotPython` performs a minimization, therefore, metrics that should be maximized have to be multiplied by -1. This is done by setting `"weights"` to `-1`.

- The complete setup for the metric in our example is:

```
from spotPython.utils.metrics import mapk_score
fun_control.update({
               "weights": -1,
               "metric_sklearn": mapk_score,
               "predict_proba": True,
               "metric_params": {"k": 3},
               })
```

17.7.2 Evaluation on Hold-out Data

- The default method for computing the performance is `"eval_holdout"`.
- Alternatively, cross-validation can be used for every machine learning model.
- Specifically for RandomForests, the OOB-score can be used.

```
fun_control.update({
    "eval": "train_hold_out",
})
```

17.7.2.1 Cross Validation

Instead of using the OOB-score, the classical cross validation can be used. The number of folds is set by the key `"k_folds"`. For example, to use 5-fold cross validation, the key `"k_folds"` is set to `5`. Uncomment the following line to use cross validation:

```python
# fun_control.update({
#      "eval": "train_cv",
#      "k_folds": 10,
# })
```

17.8 Step 8: Calling the SPOT Function

17.8.1 Preparing the SPOT Call

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

```python
# 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")
```

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

```
| name        | type   | default   |   lower |   upper | transform   |
|-------------|--------|-----------|---------|---------|-------------|
| C           | float  | 1.0       |  0.1    |   10    | None        |
| kernel      | factor | rbf       |  0      |    0    | None        |
| degree      | int    | 3         |  3      |    3    | None        |
| gamma       | factor | scale     |  0      |    1    | None        |
| coef0       | float  | 0.0       |  0      |    0    | None        |
| shrinking   | factor | 0         |  0      |    1    | None        |
| probability | factor | 0         |  1      |    1    | None        |
| tol         | float  | 0.001     |  0.0001 |    0.01 | None        |
| cache_size  | float  | 200.0     | 100     |  400    | None        |
| break_ties  | factor | 0         |  0      |    1    | None        |
```

17.8.2 The Objective Function

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

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

17.8.3 Run the `Spot` Optimizer

- Run SPOT for approx. x mins (`max_time`).
- Note: the run takes longer, because the evaluation time of initial design (here: `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,
```

```python
                     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: -0.37092731829573933 [----------] 0.25%

spotPython tuning: -0.3784461152882205З [----------] 0.52%

spotPython tuning: -0.37844611528822053 [----------] 0.83%

spotPython tuning: -0.37844611528822053 [----------] 1.23%

spotPython tuning: -0.38596491228070173 [----------] 1.65%

spotPython tuning: -0.38596491228070173 [----------] 2.07%

spotPython tuning: -0.38596491228070173 [----------] 2.51%

spotPython tuning: -0.38596491228070173 [----------] 3.01%

spotPython tuning: -0.38596491228070173 [----------] 3.51%

spotPython tuning: -0.38596491228070173 [----------] 4.13%

spotPython tuning: -0.38847117794486213 [----------] 4.76%

spotPython tuning: -0.38847117794486213 [#---------] 5.31%
```

```
spotPython tuning: -0.38847117794486213 [#---------] 6.74%

spotPython tuning: -0.38847117794486213 [#---------] 8.16%

spotPython tuning: -0.38847117794486213 [#---------] 9.87%

spotPython tuning: -0.38847117794486213 [#---------] 11.59%

spotPython tuning: -0.38847117794486213 [#---------] 12.91%

spotPython tuning: -0.38847117794486213 [#---------] 14.13%

spotPython tuning: -0.38847117794486213 [##--------] 15.40%

spotPython tuning: -0.38847117794486213 [##--------] 17.01%

spotPython tuning: -0.38847117794486213 [##--------] 18.78%

spotPython tuning: -0.38847117794486213 [##--------] 20.23%

spotPython tuning: -0.38847117794486213 [##--------] 21.47%

spotPython tuning: -0.38847117794486213 [##--------] 22.71%

spotPython tuning: -0.38847117794486213 [##--------] 23.91%

spotPython tuning: -0.38847117794486213 [###-------] 25.19%

spotPython tuning: -0.38847117794486213 [###-------] 26.69%

spotPython tuning: -0.38847117794486213 [###-------] 28.39%

spotPython tuning: -0.38847117794486213 [###-------] 30.24%

spotPython tuning: -0.38847117794486213 [###-------] 31.99%

spotPython tuning: -0.38847117794486213 [###-------] 34.00%
```

```
spotPython tuning: -0.38847117794486213 [####------] 35.79%

spotPython tuning: -0.38847117794486213 [####------] 37.80%

spotPython tuning: -0.38847117794486213 [####------] 39.91%

spotPython tuning: -0.38847117794486213 [####------] 42.34%

spotPython tuning: -0.38847117794486213 [#####-----] 45.09%

spotPython tuning: -0.38847117794486213 [#####-----] 47.85%

spotPython tuning: -0.38847117794486213 [#####-----] 50.12%

spotPython tuning: -0.38847117794486213 [#####-----] 52.32%

spotPython tuning: -0.38847117794486213 [#####-----] 54.49%

spotPython tuning: -0.38847117794486213 [######----] 56.60%

spotPython tuning: -0.38847117794486213 [######----] 58.63%

spotPython tuning: -0.38847117794486213 [######----] 60.94%

spotPython tuning: -0.38847117794486213 [######----] 63.55%

spotPython tuning: -0.38847117794486213 [#######---] 66.05%

spotPython tuning: -0.38847117794486213 [#######---] 68.20%

spotPython tuning: -0.38847117794486213 [#######---] 70.39%

spotPython tuning: -0.38847117794486213 [#######---] 73.00%

spotPython tuning: -0.38847117794486213 [########--] 75.38%

spotPython tuning: -0.38847117794486213 [########--] 77.39%
```

```
spotPython tuning: -0.38847117794486213 [########--] 79.53%

spotPython tuning: -0.38847117794486213 [########--] 81.95%

spotPython tuning: -0.38847117794486213 [########--] 84.16%

spotPython tuning: -0.38847117794486213 [#########-] 85.87%

spotPython tuning: -0.38847117794486213 [#########-] 88.11%

spotPython tuning: -0.38847117794486213 [#########-] 89.94%

spotPython tuning: -0.38847117794486213 [#########-] 92.19%

spotPython tuning: -0.38847117794486213 [#########-] 94.20%

spotPython tuning: -0.38847117794486213 [##########] 97.68%

spotPython tuning: -0.38847117794486213 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x146c1fd00>
```

17.9 Step 9: Tensorboard

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

17.10 Step 10: Results

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

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

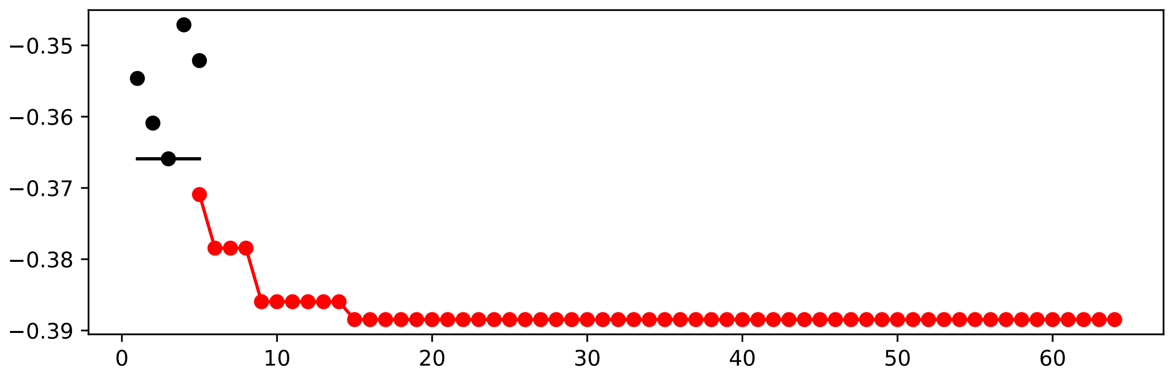


Figure 17.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- Print the results

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

```
| name        | type   | default   |   lower |   upper |                tuned | transform
|-------------|--------|-----------|---------|---------|----------------------|-------------
| C           | float  | 1.0       |     0.1 |    10.0 |   2.8512422124172496 | None
| kernel      | factor | rbf       |     0.0 |     0.0 |                  0.0 | None
| degree      | int    | 3         |     3.0 |     3.0 |                  3.0 | None
| gamma       | factor | scale     |     0.0 |     1.0 |                  1.0 | None
| coef0       | float  | 0.0       |     0.0 |     0.0 |                  0.0 | None
| shrinking   | factor | 0         |     0.0 |     1.0 |                  0.0 | None
| probability | factor | 0         |     1.0 |     1.0 |                  1.0 | None
| tol         | float  | 0.001     |  0.0001 |    0.01 | 0.005558166960224699 | None
| cache_size  | float  | 200.0     |   100.0 |   400.0 |                100.0 | None
| break_ties  | factor | 0         |     0.0 |     1.0 |                  1.0 | None
```

17.10.1 Show variable importance

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

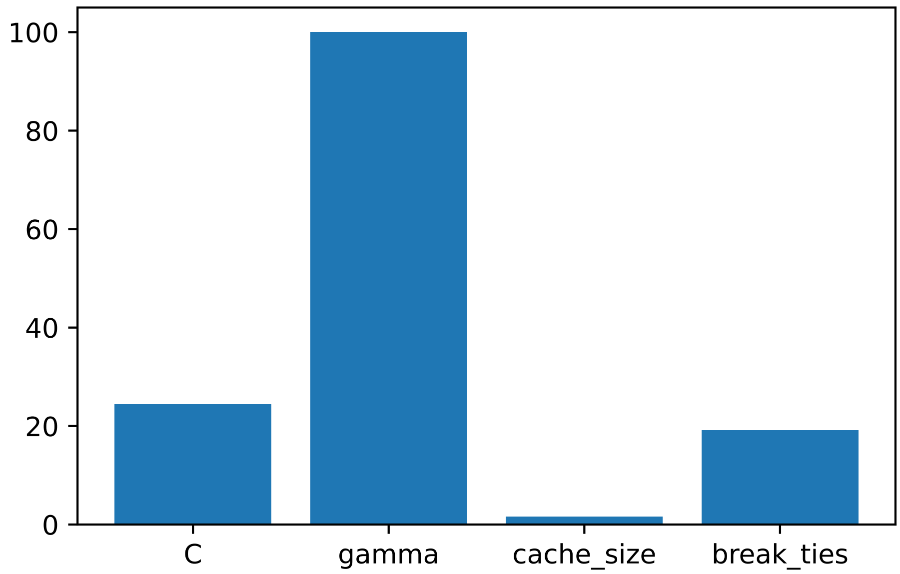


Figure 17.2: Variable importance plot, threshold 0.025.

17.10.2 Get Default Hyperparameters

```
from spotPython.hyperparameters.values import get_default_values, transform_hyper_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=[('nonetype', None),
                ('svc',
                 SVC(break_ties=0, cache_size=200.0, probability=0,
                     shrinking=0))])
```

> **Note**
>
> - Default value for "probability" is False, but we need it to be True for the metric "mapk_score".
>
> ```
> values_default.update({"probability": 1})
> ```

17.10.3 Get SPOT Results

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

```
[[2.85124221e+00 0.00000000e+00 3.00000000e+00 1.00000000e+00
  0.00000000e+00 0.00000000e+00 1.00000000e+00 5.55816696e-03
  1.00000000e+02 1.00000000e+00]]
```

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

```
[{'C': 2.8512422124172496,
  'kernel': 'rbf',
  'degree': 3,
  'gamma': 'auto',
  'coef0': 0.0,
```

```
'shrinking': 0,
'probability': 1,
'tol': 0.005558166960224699,
'cache_size': 100.0,
'break_ties': 1}]
```

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

```
SVC(C=2.8512422124172496, break_ties=1, cache_size=100.0, gamma='auto',
    probability=1, shrinking=0, tol=0.005558166960224699)
```

17.10.4 Evaluate SPOT Results

- Fetch the data.

```
from spotPython.utils.convert import get_Xy_from_df
X_train, y_train = get_Xy_from_df(fun_control["train"], fun_control["target_column"])
X_test, y_test = get_Xy_from_df(fun_control["test"], fun_control["target_column"])
X_test.shape, y_test.shape
```

```
((177, 64), (177,))
```

- Fit the model with the tuned hyperparameters. This gives one result:

```
model_spot.fit(X_train, y_train)
y_pred = model_spot.predict_proba(X_test)
res = mapk_score(y_true=y_test, y_pred=y_pred, k=3)
res
```

```
0.37099811676082856
```

```
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)
```

```python
        res = mapk_score(y_true=y_test, y_pred=y_pred, k=3)
        res_values.append(res)
    mean_res = np.mean(res_values)
    print(f"mean_res: {mean_res}")
    std_res = np.std(res_values)
    print(f"std_res: {std_res}")
    min_res = np.min(res_values)
    print(f"min_res: {min_res}")
    max_res = np.max(res_values)
    print(f"max_res: {max_res}")
    median_res = np.median(res_values)
    print(f"median_res: {median_res}")
    return mean_res, std_res, min_res, max_res, median_res
```

17.10.5 Handling Non-deterministic Results

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

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

```
mean_res: 0.37551789077212805
std_res: 0.005222382155880242
min_res: 0.3644067796610169
max_res: 0.3870056497175141
median_res: 0.3757062146892655
```

17.10.6 Evalution of the Default Hyperparameters

```python
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.

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

```
0.384180790960452
```

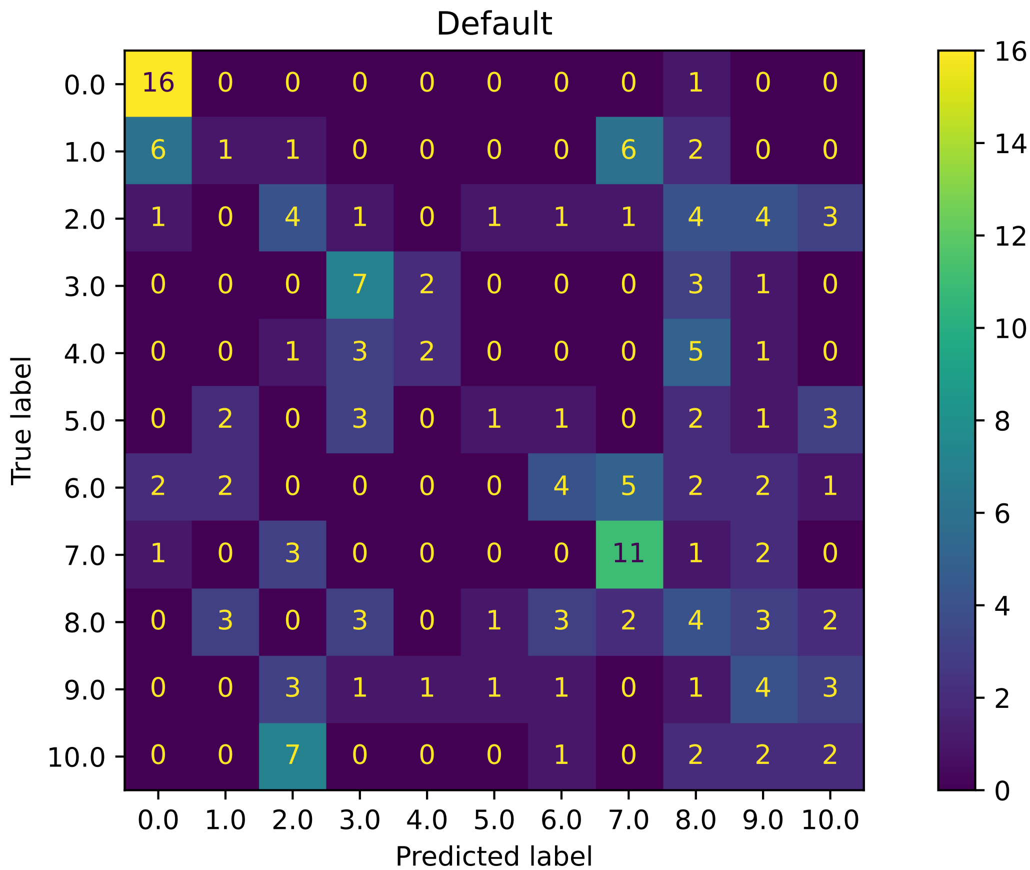
Since one single evaluation is not meaningful, we perform, similar to the evaluation of the SPOT results, $n = 30$ runs of the default setting and and calculate the mean and standard deviation of the performance metric.

```
_ = repeated_eval(30, model_default)
```

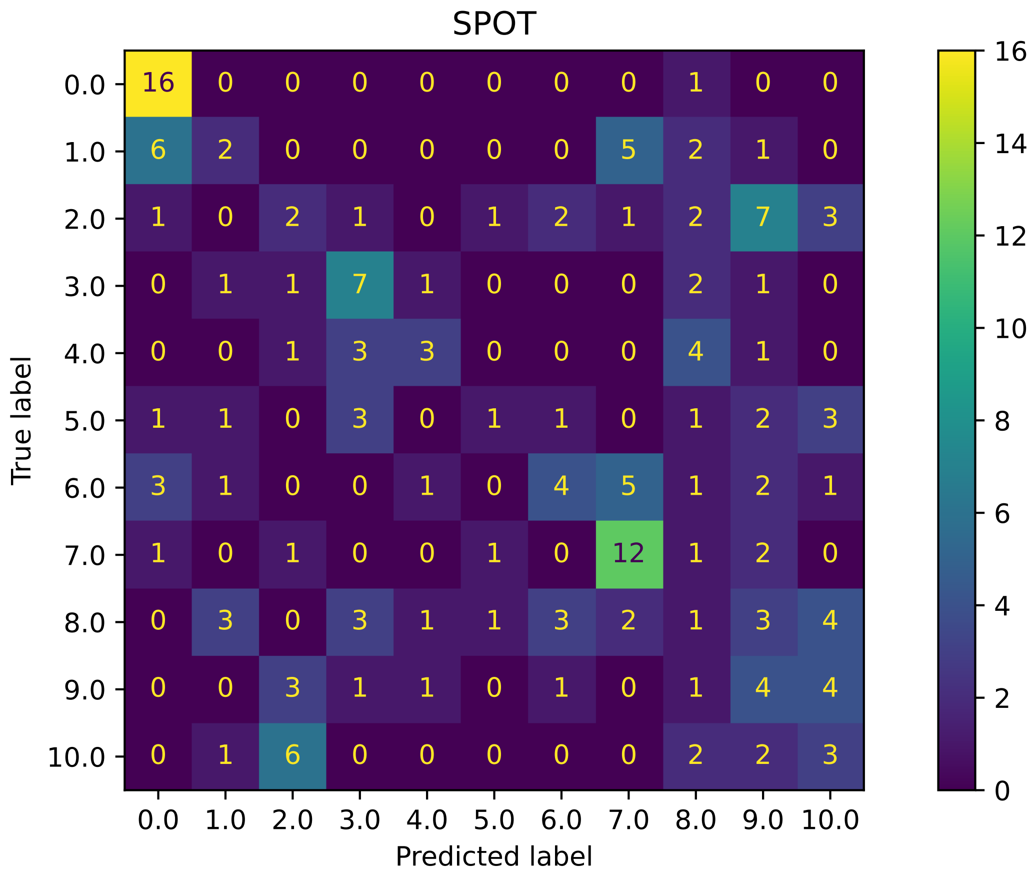
```
mean_res: 0.3847143753923414
std_res: 0.0050747578855796106
min_res: 0.37664783427495285
max_res: 0.4011299435028249
median_res: 0.38370998116760835
```

17.10.7 Plot: Compare Predictions

```
from spotPython.plot.validation import plot_confusion_matrix
plot_confusion_matrix(model_default, fun_control, title = "Default")
```




```python
plot_confusion_matrix(model_spot, fun_control, title="SPOT")
```




```
min(spot_tuner.y), max(spot_tuner.y)
```

```
(-0.38847117794486213, -0.3408521303258145)
```

17.10.8 Cross-validated Evaluations

```
from spotPython.sklearn.traintest import evaluate_cv
fun_control.update({
     "eval": "train_cv",
     "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)
```

```
(0.3578616352201258, None)
```

```python
fun_control.update({
     "eval": "test_cv",
     "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)
```

```
(0.35506535947712414, None)
```

- This is the evaluation that will be used in the comparison:

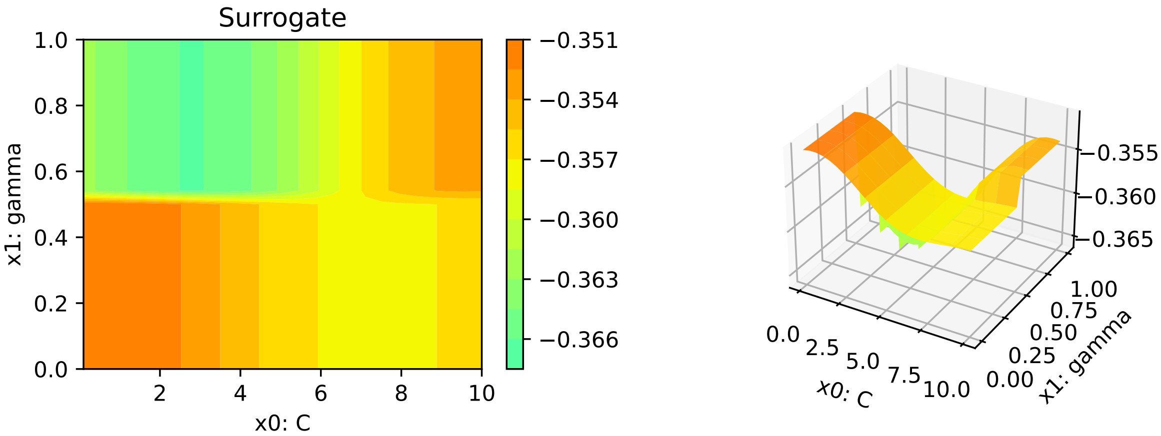
```python
fun_control.update({
     "eval": "data_cv",
     "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)
```

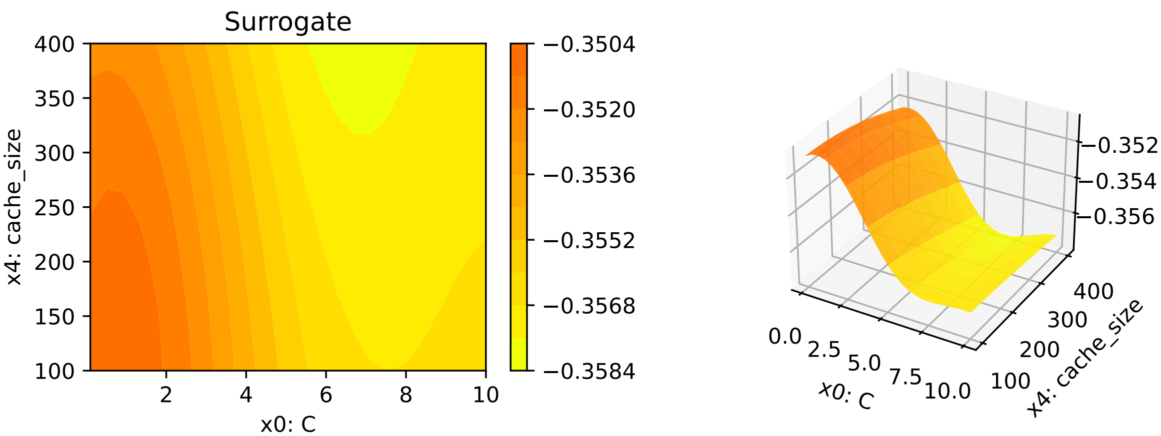
```
(0.36410462776659963, None)
```

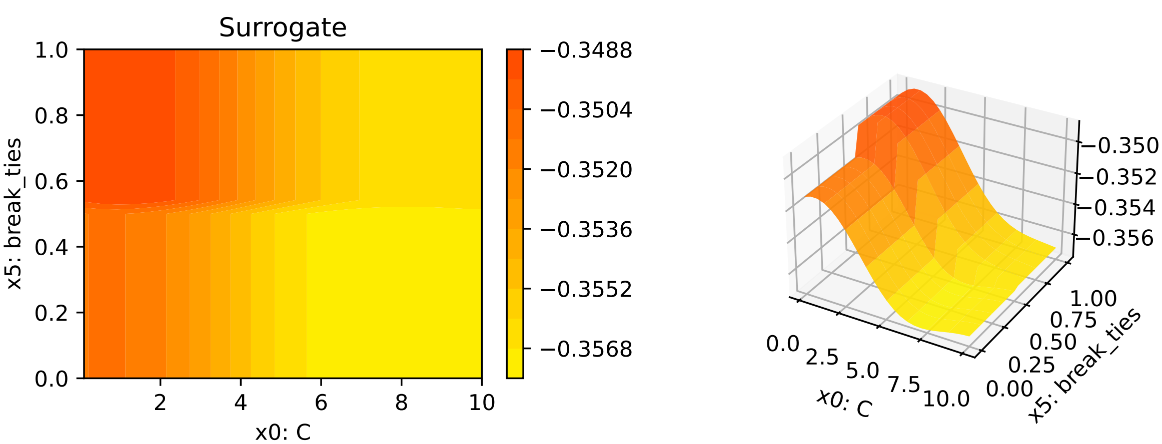
17.10.9 Detailed Hyperparameter Plots

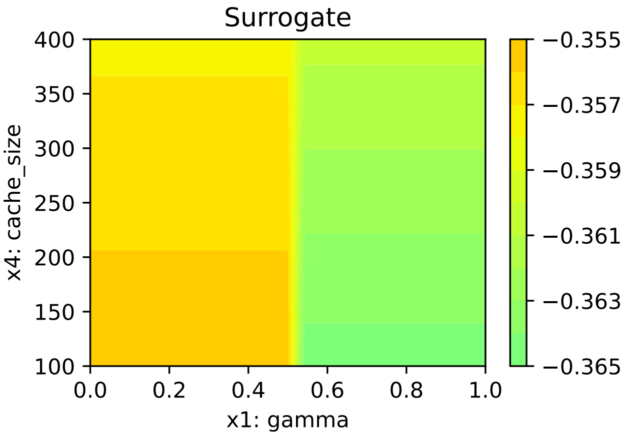
```python
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

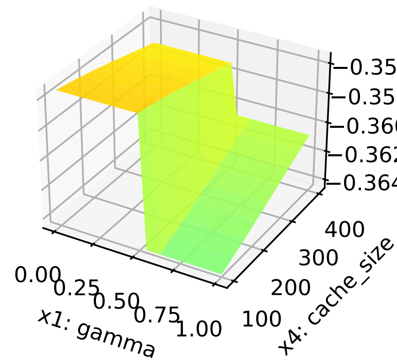
```
C:  24.35651161184802
gamma:  100.0
cache_size:  1.5596335678858653
break_ties:  19.09639342688839
```

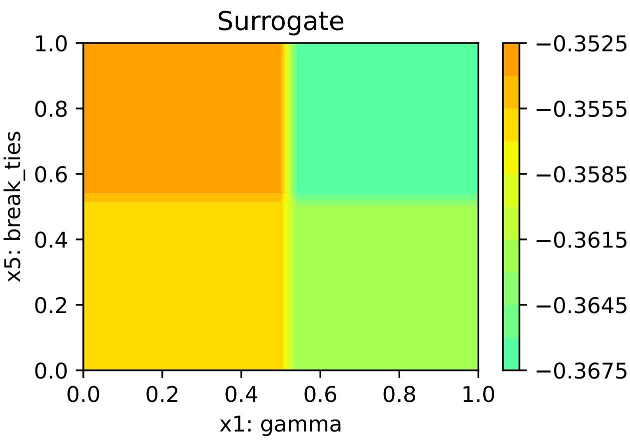

Surrogate
x1: gamma
x0: C
−0.351
−0.354
−0.357
−0.360
−0.363
−0.366
x0: C
x1: gamma
−0.355
−0.360
−0.365

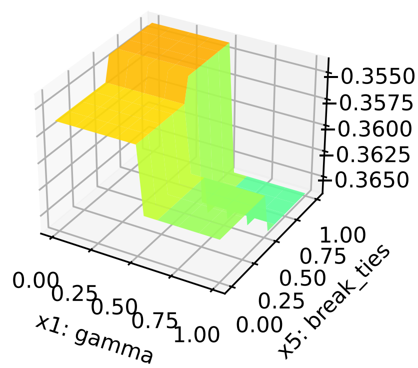

Surrogate
x4: cache_size
x0: C
−0.3504
−0.3520
−0.3536
−0.3552
−0.3568
−0.3584
x0: C
x4: cache_size
−0.352
−0.354
−0.356

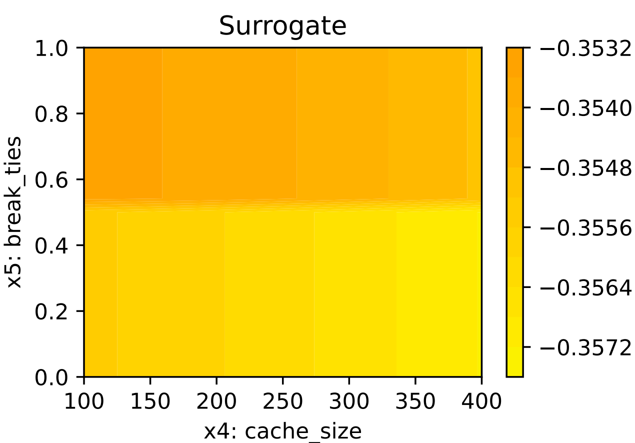

Surrogate
x5: break_ties
x0: C
−0.3488
−0.3504
−0.3520
−0.3536
−0.3552
−0.3568
x0: C
x5: break_ties
−0.350
−0.352
−0.354
−0.356

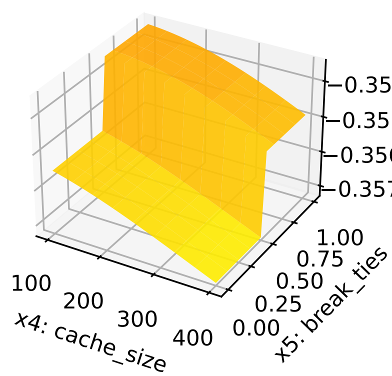

Surrogate
x4: cache_size
400
350
300
250
200
150
100
0.0
0.2
0.4
0.6
0.8
1.0
x1: gamma
−0.355
−0.357
−0.359
−0.361
−0.363
−0.365


−0.356
−0.358
−0.360
−0.362
−0.364
0.00
0.25
0.50
0.75
1.00
x1: gamma
100
200
300
400
x4: cache_size


Surrogate
x5: break_ties
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
x1: gamma
−0.3525
−0.3555
−0.3585
−0.3615
−0.3645
−0.3675


−0.3550
−0.3575
−0.3600
−0.3625
−0.3650
0.00
0.25
0.50
0.75
1.00
x1: gamma
0.00
0.25
0.50
0.75
1.00
x5: break_ties


Surrogate
x5: break_ties
1.0
0.8
0.6
0.4
0.2
0.0
100
150
200
250
300
350
400
x4: cache_size
−0.3532
−0.3540
−0.3548
−0.3556
−0.3564
−0.3572


−0.354
−0.355
−0.356
−0.357
100
200
300
400
x4: cache_size
0.00
0.25
0.50
0.75
1.00
x5: break_ties

17.10.10 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): text/html
```

17.10.11 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

```
PLOT_ALL = False
if PLOT_ALL:
    n = spot_tuner.k
    for i in range(n-1):
        for j in range(i+1, n):
            spot_tuner.plot_contour(i=i, j=j, min_z=min_z, max_z = max_z)
```

18 HPT: sklearn KNN Classifier VBDP Data

This document refers to the following software versions:

- `python`: 3.10.10

```
pip list | grep  "spot[RiverPython]"
```

```
spotPython                    0.2.46
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
```

18.1 Step 1: Setup

Before we consider the detailed experimental setup, we select the parameters that affect run time and the initial design size.

```
MAX_TIME = 1
INIT_SIZE = 5
ORIGINAL = False
```

```
import os
import copy
import socket
from datetime import datetime
from dateutil.tz import tzlocal
start_time = datetime.now(tzlocal())
HOSTNAME = socket.gethostname().split(".")[0]
experiment_name = '19-knn-sklearn' + "_" + HOSTNAME + "_" + str(MAX_TIME) + "min_" + str(I
experiment_name = experiment_name.replace(':', '-')
print(experiment_name)
if not os.path.exists('./figures'):
    os.makedirs('./figures')
```

```
19-knn-sklearn_bartz09_1min_5init_2023-06-27_04-04-21
```

```
import warnings
warnings.filterwarnings("ignore")
```

18.2 Step 2: Initialization of the Empty `fun_control` Dictionary

> Caution: Tensorboard does not work under Windows
>
> - Since tensorboard does not work under Windows, we recommend setting the parameter `tensorboard_path` to `None` if you are working under Windows.

```
from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="classification",
    tensorboard_path="runs/16_spot_hpt_sklearn_classification")
```

18.2.1 Load Data: Classification VBDP

```
import pandas as pd
if ORIGINAL == True:
    train_df = pd.read_csv('./data/VBDP/trainn.csv')
    test_df = pd.read_csv('./data/VBDP/testt.csv')
else:
    train_df = pd.read_csv('./data/VBDP/train.csv')
```

```
    # remove the id column
    train_df = train_df.drop(columns=['id'])
```

```
from sklearn.preprocessing import OrdinalEncoder
n_samples = train_df.shape[0]
n_features = train_df.shape[1] - 1
target_column = "prognosis"
# Encoder our prognosis labels as integers for easier decoding later
enc = OrdinalEncoder()
train_df[target_column] = enc.fit_transform(train_df[[target_column]])
train_df.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
print(train_df.shape)
train_df.head()
```

```
(707, 65)
```

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	...	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`.

18.2.2 Holdout Train and Test Data

We split out a hold-out test set (25% of the data) so we can calculate an example MAP@K

```
import numpy as np
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(train_df.drop(target_column, axis=1),
                                                    random_state=42,
                                                    test_size=0.25,
                                                    stratify=train_df[target_column])
train = pd.DataFrame(np.hstack((X_train, np.array(y_train).reshape(-1, 1))))
test = pd.DataFrame(np.hstack((X_test, np.array(y_test).reshape(-1, 1))))
train.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
```

```python
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

```python
# add the dataset to the fun_control
fun_control.update({"data": train_df, # full dataset,
               "train": train,
               "test": test,
               "n_samples": n_samples,
               "target_column": target_column})
```

18.3 Step 4: Specification of the Preprocessing Model

Data preprocesssing can be very simple, e.g., you can ignore it. Then you would choose the `prep_model` “None”:

```python
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:

```python
# prep_model = StandardScaler()
# fun_control.update({"prep_model": prep_model})
```

Even more complicated pre-processing steps are possible, e.g., the follwing pipeline:

```python
# categorical_columns = []
# one_hot_encoder = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
# prep_model = ColumnTransformer(
#         transformers=[
#             ("categorical", one_hot_encoder, categorical_columns),
#         ],
#         remainder=StandardScaler(),
#     )
```

18.4 Step 5: Select Model (`algorithm`) and `core_model_hyper_dict`

The selection of the algorithm (ML model) that should be tuned is done by specifying the its name from the `sklearn` implementation. For example, the `SVC` support vector machine classifier is selected as follows:

`fun_control = add_core_model_to_fun_control(SVC, fun_control, SklearnHyperDict)`

Other core_models are, e.g.,:

- RidgeCV
- GradientBoostingRegressor
- ElasticNet
- RandomForestClassifier
- LogisticRegression
- KNeighborsClassifier
- RandomForestClassifier
- GradientBoostingClassifier
- HistGradientBoostingClassifier

We will use the `RandomForestClassifier` classifier in this example.

```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
```

```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
```

18.5 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model`

18.5.1 Modify hyperparameter of type numeric and integer (boolean)

Numeric and boolean values can be modified using the `modify_hyper_parameter_bounds` method. For example, to change the `tol` hyperparameter of the `SVC` model to the interval [1e-3, 1e-2], the following code can be used:

`fun_control = modify_hyper_parameter_bounds(fun_control, "tol", bounds=[1e-3, 1e-2])`

```
# from spotPython.hyperparameters.values import modify_hyper_parameter_bounds
# fun_control = modify_hyper_parameter_bounds(fun_control, "probability", bounds=[1, 1])
```

18.5.2 Modify hyperparameter of type factor

`spotPython` provides functions for modifying the hyperparameters, their bounds and factors as well as for activating and de-activating hyperparameters without re-compilation of the Python source code. These functions were described in Section 14.6.

Factors can be modified with the `modify_hyper_parameter_levels` function. For example, to exclude the `sigmoid` kernel from the tuning, the `kernel` hyperparameter of the `SVC` model can be modified as follows:

`fun_control = modify_hyper_parameter_levels(fun_control, "kernel", ["linear", "rbf"])`

The new setting can be controlled via:

`fun_control["core_model_hyper_dict"]["kernel"]`

```
# from spotPython.hyperparameters.values import modify_hyper_parameter_levels
# fun_control = modify_hyper_parameter_levels(fun_control, "kernel", ["rbf"])
```

18.5.3 Optimizers

Optimizers are described in Section 14.6.1.

18.5.4 Selection of the Objective: Metric and Loss Functions

- Machine learning models are optimized with respect to a metric, for example, the `accuracy` function.
- Deep learning, e.g., neural networks are optimized with respect to a loss function, for example, the `cross_entropy` function and evaluated with respect to a metric, for example, the `accuracy` function.

18.6 Step 7: Selection of the Objective (Loss) Function

The loss function, that is usually used in deep learning for optimizing the weights of the net, is stored in the `fun_control` dictionary as `"loss_function"`.

18.6.1 Metric Function

There are two different types of metrics in `spotPython`:

1. `"metric_river"` is used for the river based evaluation via `eval_oml_iter_progressive`.
2. `"metric_sklearn"` is used for the sklearn based evaluation.

We will consider multi-class classification metrics, e.g., `mapk_score` and `top_k_accuracy_score`.

> **Predict Probabilities**
>
> In this multi-class classification example the machine learning algorithm should return the probabilities of the specific classes (`"predict_proba"`) instead of the predicted values.

We set `"predict_proba"` to `True` in the `fun_control` dictionary.

18.6.1.1 The MAPK Metric

To select the MAPK metric, the following two entries can be added to the `fun_control` dictionary:

`"metric_sklearn": mapk_score"`

`"metric_params": {"k": 3}`.

18.6.1.2 Other Metrics

Alternatively, other metrics for multi-class classification can be used, e.g.,: * top_k_accuracy_score or * roc_auc_score

The metric `roc_auc_score` requires the parameter `"multi_class"`, e.g.,

`"multi_class": "ovr"`.

This is set in the `fun_control` dictionary.

> **Weights**
>
> `spotPython` performs a minimization, therefore, metrics that should be maximized have to be multiplied by -1. This is done by setting `"weights"` to `-1`.

- The complete setup for the metric in our example is:

```python
from spotPython.utils.metrics import mapk_score
fun_control.update({
               "weights": -1,
               "metric_sklearn": mapk_score,
               "predict_proba": True,
               "metric_params": {"k": 3},
               })
```

18.6.2 Evaluation on Hold-out Data

- The default method for computing the performance is `"eval_holdout"`.
- Alternatively, cross-validation can be used for every machine learning model.
- Specifically for RandomForests, the OOB-score can be used.

```python
fun_control.update({
    "eval": "train_hold_out",
})
```

18.6.2.1 Cross Validation

Instead of using the OOB-score, the classical cross validation can be used. The number of folds is set by the key `"k_folds"`. For example, to use 5-fold cross validation, the key `"k_folds"` is set to `5`. Uncomment the following line to use cross validation:

```python
# fun_control.update({
#      "eval": "train_cv",
#      "k_folds": 10,
# })
```

18.7 Step 8: Calling the SPOT Function

18.7.1 Preparing the SPOT Call

- Get types and variable names as well as lower and upper bounds for the hyperparameters.

```python
# extract the variable types, names, and bounds
from spotPython.hyperparameters.values import (get_bound_values,
    get_var_name,
```

```
    get_var_type,)
var_type = get_var_type(fun_control)
var_name = get_var_name(fun_control)
fun_control.update({"var_type": var_type,
                    "var_name": var_name})
lower = get_bound_values(fun_control, "lower")
upper = get_bound_values(fun_control, "upper")
```

```
from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control))
```

```
| name        | type   | default   |   lower |   upper | transform              |
|-------------|--------|-----------|---------|---------|------------------------|
| n_neighbors | int    | 2         |       1 |       7 | transform_power_2_int |
| weights     | factor | uniform   |       0 |       1 | None                   |
| algorithm   | factor | auto      |       0 |       3 | None                   |
| leaf_size   | int    | 5         |       2 |       7 | transform_power_2_int |
| p           | int    | 2         |       1 |       2 | None                   |
```

18.7.2 The Objective Function

The objective function is selected next. It implements an interface from `sklearn`'s training, validation, and testing methods to `spotPython`.

```
from spotPython.fun.hypersklearn import HyperSklearn
fun = HyperSklearn().fun_sklearn
```

18.7.3 Run the `Spot` Optimizer

- Run SPOT for approx. x mins (`max_time`).
- Note: the run takes longer, because the evaluation time of initial design (here: `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 [----------] 0.52%

spotPython tuning: -0.3107769423558897 [----------] 0.78%

spotPython tuning: -0.3107769423558897 [----------] 1.02%

spotPython tuning: -0.3107769423558897 [----------] 1.28%
```

```
spotPython tuning: -0.3107769423558897 [----------] 1.59%

spotPython tuning: -0.3107769423558897 [----------] 1.94%

spotPython tuning: -0.3107769423558897 [----------] 2.27%

spotPython tuning: -0.3107769423558897 [----------] 2.59%

spotPython tuning: -0.3107769423558897 [----------] 2.88%

spotPython tuning: -0.3107769423558897 [----------] 3.18%

spotPython tuning: -0.3107769423558897 [----------] 4.15%

spotPython tuning: -0.3107769423558897 [#---------] 5.17%

spotPython tuning: -0.3107769423558897 [#---------] 6.35%

spotPython tuning: -0.3107769423558897 [#---------] 7.54%

spotPython tuning: -0.3107769423558897 [#---------] 8.79%

spotPython tuning: -0.3107769423558897 [#---------] 10.26%

spotPython tuning: -0.3107769423558897 [#---------] 11.43%

spotPython tuning: -0.3107769423558897 [#---------] 13.52%

spotPython tuning: -0.3107769423558897 [#---------] 14.91%

spotPython tuning: -0.3107769423558897 [##--------] 16.04%

spotPython tuning: -0.3107769423558897 [##--------] 16.97%

spotPython tuning: -0.3107769423558897 [##--------] 17.99%

spotPython tuning: -0.3107769423558897 [##--------] 19.23%
```

```
spotPython tuning: -0.3107769423558897 [##--------] 20.21%

spotPython tuning: -0.3107769423558897 [##--------] 21.40%

spotPython tuning: -0.3107769423558897 [##--------] 22.75%

spotPython tuning: -0.3107769423558897 [##--------] 24.25%

spotPython tuning: -0.3107769423558897 [###-------] 25.93%

spotPython tuning: -0.3107769423558897 [###-------] 27.71%

spotPython tuning: -0.3107769423558897 [###-------] 29.39%

spotPython tuning: -0.3107769423558897 [###-------] 31.75%

spotPython tuning: -0.3107769423558897 [###-------] 33.82%

spotPython tuning: -0.3107769423558897 [####------] 36.38%

spotPython tuning: -0.3107769423558897 [####------] 40.35%

spotPython tuning: -0.3107769423558897 [####------] 43.44%

spotPython tuning: -0.3107769423558897 [#####-----] 46.56%

spotPython tuning: -0.3107769423558897 [#####-----] 49.69%

spotPython tuning: -0.3107769423558897 [#####-----] 52.40%

spotPython tuning: -0.3107769423558897 [######----] 55.16%

spotPython tuning: -0.3107769423558897 [######----] 58.15%

spotPython tuning: -0.3107769423558897 [######----] 61.16%

spotPython tuning: -0.3107769423558897 [######----] 64.47%
```

```
spotPython tuning: -0.3107769423558897 [#######---] 67.71%

spotPython tuning: -0.3107769423558897 [#######---] 70.62%

spotPython tuning: -0.3107769423558897 [#######---] 73.82%

spotPython tuning: -0.3107769423558897 [########--] 78.22%

spotPython tuning: -0.3107769423558897 [########--] 81.50%

spotPython tuning: -0.3107769423558897 [########--] 84.77%

spotPython tuning: -0.3107769423558897 [#########-] 88.34%

spotPython tuning: -0.3107769423558897 [#########-] 92.27%

spotPython tuning: -0.3107769423558897 [##########] 95.54%

spotPython tuning: -0.3107769423558897 [##########] 98.03%

spotPython tuning: -0.3107769423558897 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x17f7dbd90>
```

18.8 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section 14.9, see also the description in the documentation: Tensorboard.

18.9 Step 10: Results

After the hyperparameter tuning run is finished, the progress of the hyperparameter tuning can be visualized. The following code generates the progress plot from **?@fig-progress**.

```
spot_tuner.plot_progress(log_y=False,
    filename="./figures/" + experiment_name+"_progress.png")
```

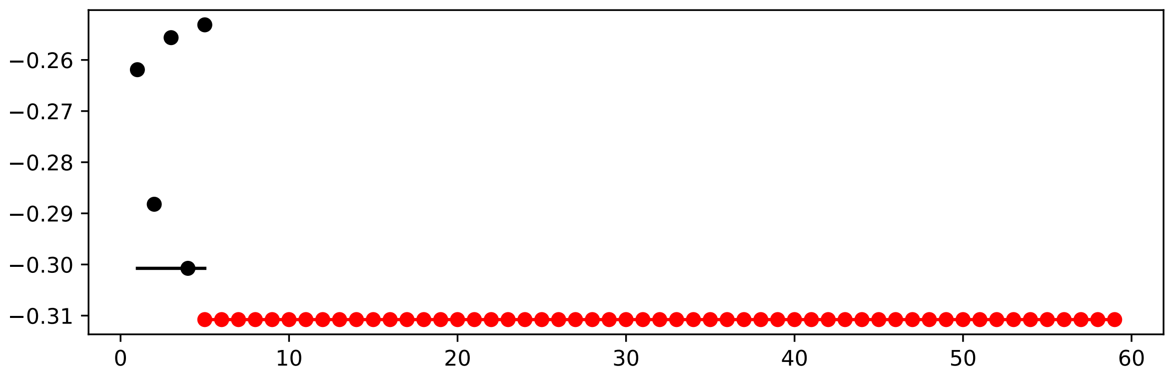


Figure 18.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

- Print the results

```
print(gen_design_table(fun_control=fun_control,
    spot=spot_tuner))
```

```
| name        | type   | default   |   lower |   upper |   tuned | transform              |
|-------------|--------|-----------|---------|---------|---------|------------------------|--
| n_neighbors | int    | 2         |       1 |       7 |     4.0 | transform_power_2_int |
| weights     | factor | uniform   |       0 |       1 |     1.0 | None                   |
| algorithm   | factor | auto      |       0 |       3 |     2.0 | None                   |
| leaf_size   | int    | 5         |       2 |       7 |     6.0 | transform_power_2_int |
| p           | int    | 2         |       1 |       2 |     1.0 | None                   |
```

18.9.1 Show variable importance

```
spot_tuner.plot_importance(threshold=0.025, filename="./figures/" + experiment_name+"_impo
```

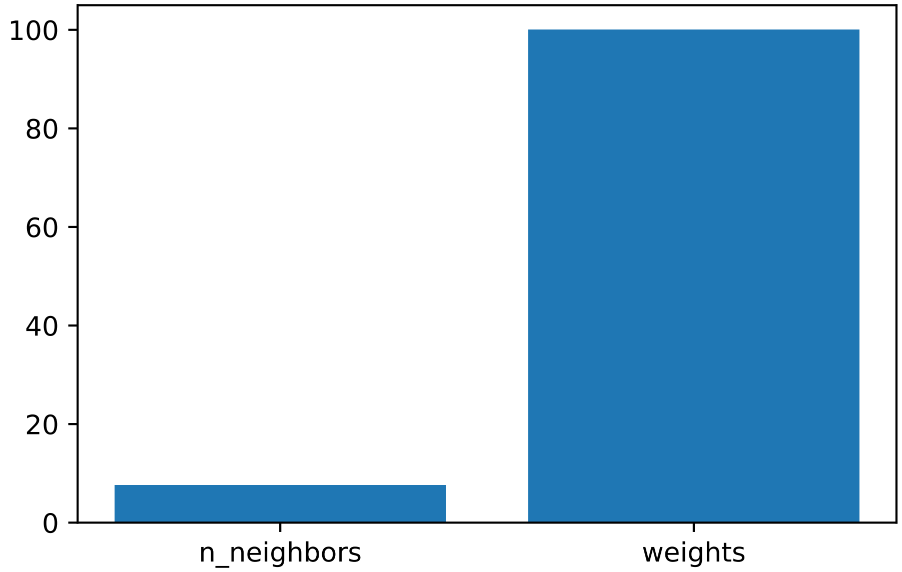


Figure 18.2: Variable importance plot, threshold 0.025.

18.9.2 Get Default Hyperparameters

```
from spotPython.hyperparameters.values import get_default_values, transform_hyper_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))])
```

18.9.3 Get SPOT Results

```
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
print(X)
```

```
[[4. 1. 2. 6. 1.]]
```

```
from spotPython.hyperparameters.values import assign_values, return_conf_list_from_var_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')
```

18.9.4 Evaluate SPOT Results

- Fetch the data.

```
from spotPython.utils.convert import get_Xy_from_df
X_train, y_train = get_Xy_from_df(fun_control["train"], fun_control["target_column"])
X_test, y_test = get_Xy_from_df(fun_control["test"], fun_control["target_column"])
X_test.shape, y_test.shape
```

```
((177, 64), (177,))
```

- Fit the model with the tuned hyperparameters. This gives one result:

```
model_spot.fit(X_train, y_train)
y_pred = model_spot.predict_proba(X_test)
res = mapk_score(y_true=y_test, y_pred=y_pred, k=3)
res
```

```
0.3267419962335216
```

```
def repeated_eval(n, model):
    res_values = []
    for i in range(n):
        model.fit(X_train, y_train)
        y_pred = model.predict_proba(X_test)
        res = mapk_score(y_true=y_test, y_pred=y_pred, k=3)
        res_values.append(res)
    mean_res = np.mean(res_values)
    print(f"mean_res: {mean_res}")
    std_res = np.std(res_values)
    print(f"std_res: {std_res}")
    min_res = np.min(res_values)
    print(f"min_res: {min_res}")
    max_res = np.max(res_values)
    print(f"max_res: {max_res}")
    median_res = np.median(res_values)
    print(f"median_res: {median_res}")
    return mean_res, std_res, min_res, max_res, median_res
```

18.9.5 Handling Non-deterministic Results

- Because the model is non-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
```

18.9.6 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
```

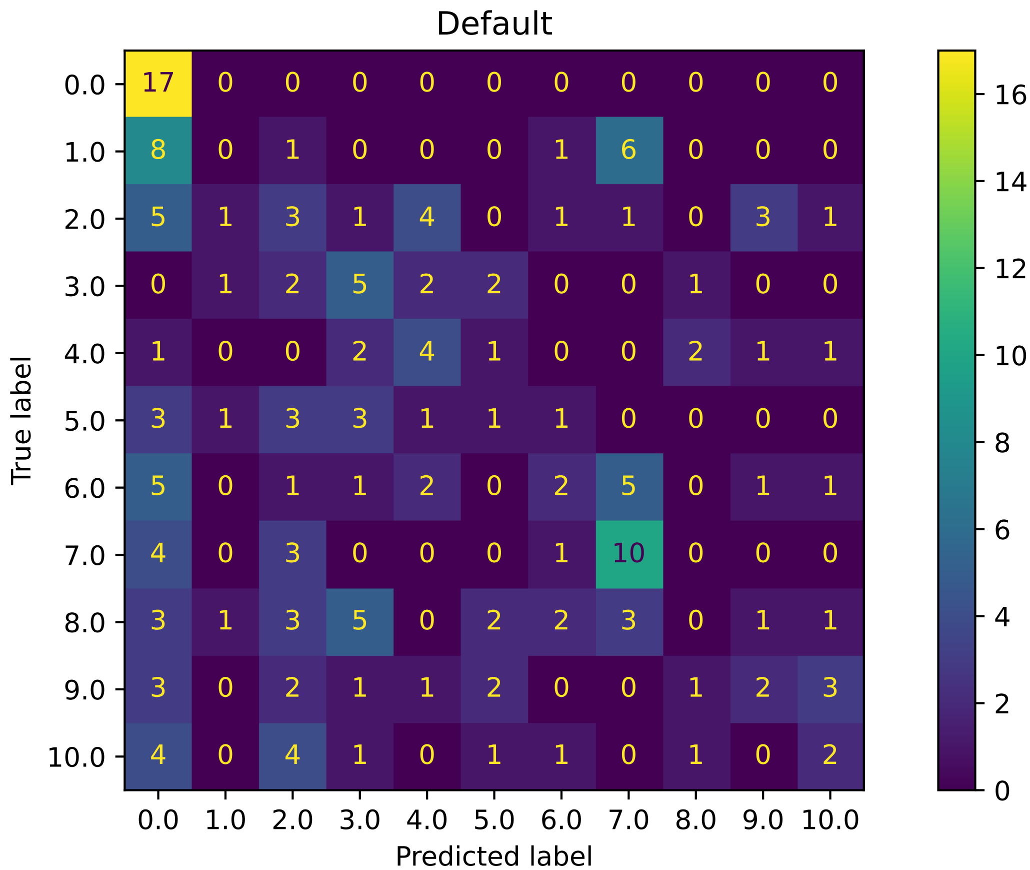
Since one single evaluation is not meaningful, we perform, similar to the evaluation of the SPOT results, $n = 30$ runs of the default setting and and calculate the mean and standard deviation of the performance metric.

```
_ = repeated_eval(30, model_default)
```

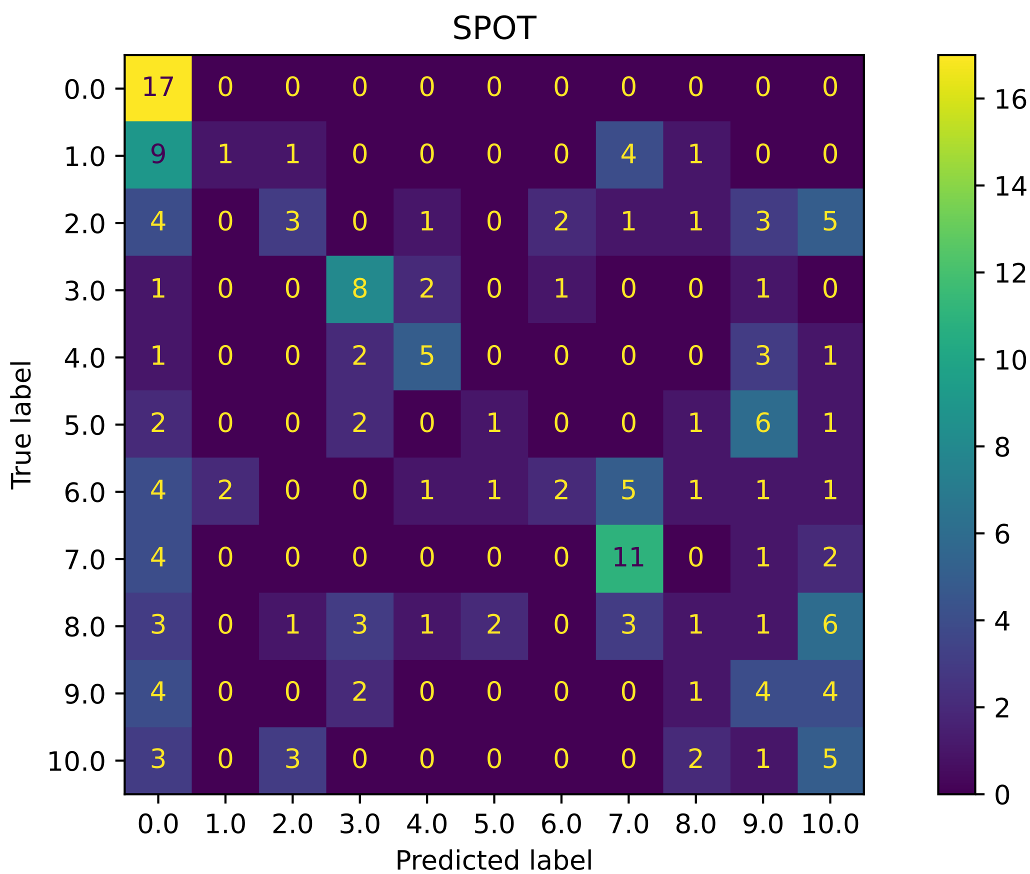
```
mean_res: 0.2768361581920903
std_res: 1.1102230246251565e-16
min_res: 0.2768361581920904
max_res: 0.2768361581920904
median_res: 0.2768361581920904
```

18.9.7 Plot: Compare Predictions

```
from spotPython.plot.validation import plot_confusion_matrix
plot_confusion_matrix(model_default, fun_control, title = "Default")
```




```
plot_confusion_matrix(model_spot, fun_control, title="SPOT")
```




```
min(spot_tuner.y), max(spot_tuner.y)
```

```
(-0.3107769423558897, -0.23558897243107768)
```

18.9.8 Cross-validated Evaluations

```
from spotPython.sklearn.traintest import evaluate_cv
fun_control.update({
     "eval": "train_cv",
     "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)
```

```
(0.3157232704402516, None)
```

```
fun_control.update({
     "eval": "test_cv",
     "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)
```

```
(0.2832788671023965, None)
```

- This is the evaluation that will be used in the comparison:

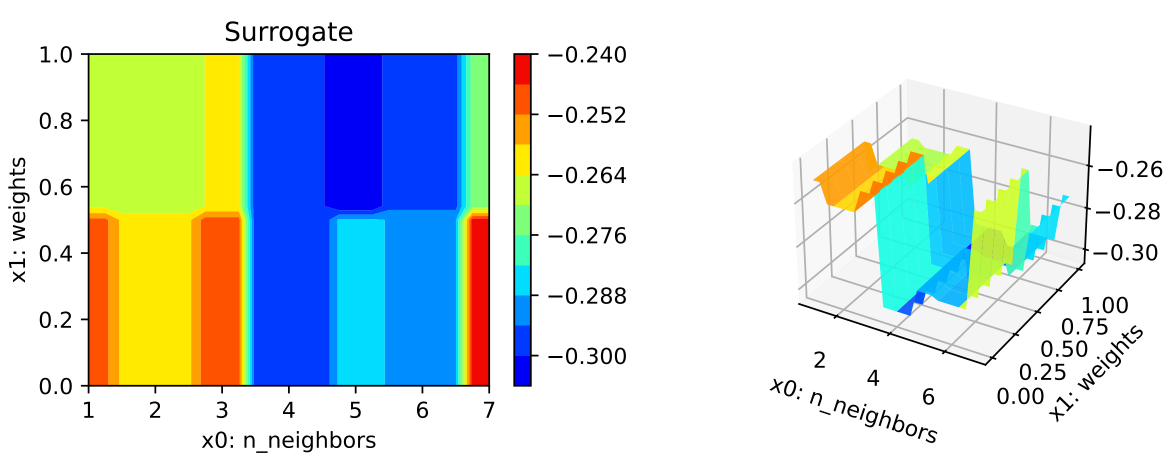
```
fun_control.update({
     "eval": "data_cv",
     "k_folds": 10,
})
evaluate_cv(model=model_spot, fun_control=fun_control, verbose=0)
```

```
(0.3061904761904762, None)
```

18.9.9 Detailed Hyperparameter Plots

```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

```
n_neighbors:  7.659298853276286
weights:  100.0
```



18.9.10 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): text/html
```

18.9.11 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

```
PLOT_ALL = False
if PLOT_ALL:
    n = spot_tuner.k
    for i in range(n-1):
        for j in range(i+1, n):
            spot_tuner.plot_contour(i=i, j=j, min_z=min_z, max_z = max_z)
```

19 HPT PyTorch: Regression

In this tutorial, we will show how `spotPython` can be integrated into the `PyTorch` training workflow for 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                    0.2.46
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
```

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 = 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_2023-06-27_04-09-21
```

19.2 Step 2: Initialization of the `fun_control` Dictionary

> Caution: Tensorboard does not work under Windows
>
> - Since tensorboard does not work under Windows, we recommend setting the parameter `tensorboard_path` to `None` if you are working under Windows.

`spotPython` uses a Python dictionary for storing the information required for the hyperparameter tuning process, which was described in Section 14.2.

```
from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="regression",
 tensorboard_path="runs/24_spot_torch_regression",
 device=DEVICE)
```

19.3 Step 3: PyTorch Data Loading

```
# Create dataset
import pandas as pd
import numpy as np
from sklearn import datasets as sklearn_datasets
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
X, y = sklearn_datasets.make_regression(
    n_samples=1000, n_features=10, noise=1, random_state=123)
y = y.reshape(-1, 1)

# Normalize the data
X_scaler = MinMaxScaler()
X_scaled = X_scaler.fit_transform(X)
y_scaler = MinMaxScaler()
y_scaled = y_scaler.fit_transform(y)

# combine the features and target into a single dataframe named train_df
train_df = pd.DataFrame(np.hstack((X_scaled, y_scaled)))

target_column = "y"
n_samples = train_df.shape[0]
n_features = train_df.shape[1] - 1
```

```
train_df.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
X_train, X_test, y_train, y_test = train_test_split(train_df.drop(target_column,
    axis=1),
    train_df[target_column],
    random_state=42,
    test_size=0.25)
trainset = pd.DataFrame(np.hstack((X_train, np.array(y_train).reshape(-1, 1))))
testset = pd.DataFrame(np.hstack((X_test, np.array(y_test).reshape(-1, 1))))
trainset.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
testset.columns = [f"x{i}" for i in range(1, n_features+1)] + [target_column]
print(train_df.shape)
print(trainset.shape)
print(testset.shape)
```

```
(1000, 11)
(750, 11)
(250, 11)
```

```
import torch
from spotPython.torch.dataframedataset import DataFrameDataset
dtype_x = torch.float32
dtype_y = torch.float32
train_df = DataFrameDataset(train_df, target_column=target_column,
    dtype_x=dtype_x, dtype_y=dtype_y)
train = DataFrameDataset(trainset, target_column=target_column,
    dtype_x=dtype_x, dtype_y=dtype_y)
test = DataFrameDataset(testset, target_column=target_column,
    dtype_x=dtype_x, dtype_y=dtype_y)
n_samples = len(train)
```

- Now we can test the data loading:

```
from spotPython.torch.traintest import create_train_val_data_loaders
trainloader, testloader = create_train_val_data_loaders(train, 2, True, 0)
for i, data in enumerate(trainloader, 0):
    inputs, labels = data
    print(inputs.shape)
    print(labels.shape)
    print(inputs)
    print(labels)
    break
```

```
torch.Size([2, 10])
torch.Size([2])
tensor([[0.7097, 0.6017, 0.7971, 0.2858, 0.4007, 0.6272, 0.6043, 0.2522, 0.4143,
         0.7312],
        [0.6289, 0.4257, 0.7041, 0.6581, 0.3342, 0.4588, 0.3366, 0.3787, 0.6587,
         0.5475]])
tensor([0.6269, 0.3924])
```

- Since this works fine, we can add the data loading to the `fun_control` dictionary:

```python
# add the dataset to the fun_control
fun_control.update({"data": train_df, # full dataset,
               "train": train,
               "test": test,
               "n_samples": n_samples,
               "target_column": target_column,})
```

19.4 Step 4: Specification of the Preprocessing Model

After the training and test data are specified and added to the `fun_control` dictionary, `spotPython` allows the specification of a data preprocessing pipeline, e.g., for the scaling of the data or for the one-hot encoding of categorical variables, see Section 14.4. This feature is not used here, so we do not change the default value (which is `None`).

19.5 Step 5: Select Model (`algorithm`) and `core_model_hyper_dict`

19.5.1 Implementing a Configurable Neural Network With spotPython

`spotPython` includes the `Net_lin_reg` class which is implemented in the file `netregression.py`.

This class inherits from the class `Net_Core` which is implemented in the file `netcore.py`, see Section 14.5.1.

```python
from torch import nn
import spotPython.torch.netcore as netcore


class Net_lin_reg(netcore.Net_Core):
    def __init__(
```

```
        self, _L_in, _L_out, l1, dropout_prob, lr_mult,
        batch_size, epochs, k_folds, patience, optimizer,
        sgd_momentum
    ):
        super(Net_lin_reg, self).__init__(
            lr_mult=lr_mult,
            batch_size=batch_size,
            epochs=epochs,
            k_folds=k_folds,
            patience=patience,
            optimizer=optimizer,
            sgd_momentum=sgd_momentum,
        )
        l2 = max(l1 // 2, 4)
        self.fc1 = nn.Linear(_L_in, l1)
        self.fc2 = nn.Linear(l1, l2)
        self.fc3 = nn.Linear(l2, _L_out)
        self.relu = nn.ReLU()
        self.softmax = nn.Softmax(dim=1)
        self.dropout1 = nn.Dropout(p=dropout_prob)
        self.dropout2 = nn.Dropout(p=dropout_prob / 2)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.dropout1(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.dropout2(x)
        x = self.fc3(x)
        return x
```

19.5.1.1 The `Net_Core` class

`Net_lin_reg` inherits from the class `Net_Core` which is implemented in the file `netcore.py`. This class was described in Section 14.5.1.

```
from spotPython.torch.netregression import Net_lin_reg
from spotPython.data.torch_hyper_dict import TorchHyperDict
from spotPython.hyperparameters.values import add_core_model_to_fun_control
fun_control = add_core_model_to_fun_control(core_model=Net_lin_reg,
```

```
                                    fun_control=fun_control,
                                    hyper_dict=TorchHyperDict,
                                    filename=None)
```

19.5.2 The Search Space

19.5.3 Configuring the Search Space With spotPython

19.5.3.1 The `hyper_dict` Hyperparameters for the Selected Algorithm

`spotPython` uses `JSON` files for the specification of the hyperparameters, which were described in Section 14.5.5.

The corresponding entries for the `core_model` class are shown below.

```
fun_control['core_model_hyper_dict']
```

```
{'_L_in': {'type': 'int',
  'default': 10,
  'transform': 'None',
  'lower': 10,
  'upper': 10},
 '_L_out': {'type': 'int',
  'default': 1,
  'transform': 'None',
  'lower': 1,
  'upper': 1},
 'l1': {'type': 'int',
  'default': 3,
  'transform': 'transform_power_2_int',
  'lower': 3,
  'upper': 8},
 'dropout_prob': {'type': 'float',
  'default': 0.01,
  'transform': 'None',
  'lower': 0.0,
  'upper': 0.9},
 'lr_mult': {'type': 'float',
  'default': 1.0,
  'transform': 'None',
  'lower': 0.1,
```

```
 'upper': 10.0},
'batch_size': {'type': 'int',
 'default': 4,
 'transform': 'transform_power_2_int',
 'lower': 1,
 'upper': 4},
'epochs': {'type': 'int',
 'default': 4,
 'transform': 'transform_power_2_int',
 'lower': 4,
 'upper': 9},
'k_folds': {'type': 'int',
 'default': 1,
 'transform': 'None',
 'lower': 1,
 'upper': 1},
'patience': {'type': 'int',
 'default': 2,
 'transform': 'transform_power_2_int',
 'lower': 1,
 'upper': 5},
'optimizer': {'levels': ['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`

`spotPython` provides functions for modifying the hyperparameters, their bounds and factors as well as for activating and de-activating hyperparameters without re-compilation of the Python source code. These functions were described in Section 14.6.

19.6.1 Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model`

19.6.1.1 Modify Hyperparameters of Type numeric and integer (boolean)

```python
from spotPython.hyperparameters.values import modify_hyper_parameter_bounds

fun_control = modify_hyper_parameter_bounds(fun_control, "epochs", bounds=[2, 16])
fun_control = modify_hyper_parameter_bounds(fun_control, "patience", bounds=[3, 7])
```

19.6.1.2 Modify Hyperparameter of Type factor

```python
from spotPython.hyperparameters.values import modify_hyper_parameter_levels
fun_control = modify_hyper_parameter_levels(fun_control, "optimizer",
    ["Adadelta", "Adagrad", "Adam", "AdamW", "Adamax", "ASGD", "NAdam"])


fun_control.update({
               "_L_in": n_features,
               "_L_out": 1,})
```

19.6.2 Optimizers

Optimizers are described in Section 14.6.1.

19.7 Step 7: Selection of the Objective (Loss) Function

19.7.1 Evaluation

The evaluation procedure requires the specification of two elements:

1. the way how the data is split into a train and a test set (see Section 14.7.1)
2. the loss function (and a metric).

19.7.2 Loss Functions and Metrics

The key `"loss_function"` specifies the loss function which is used during the optimization, see Section 14.7.5.

We will use MSE loss for the regression task.

```
from torch.nn import MSELoss
loss_torch = MSELoss()
fun_control.update({"loss_function": loss_torch})
```

19.7.3 Metric

```
from torchmetrics import MeanAbsoluteError
metric_torch = MeanAbsoluteError().to(fun_control["device"])
fun_control.update({"metric_torch": metric_torch})
```

19.8 Step 8: Calling the SPOT Function

19.8.1 Preparing the SPOT Call

The following code passes the information about the parameter ranges and bounds to `spot`.

```
# extract the variable types, names, and bounds
from spotPython.hyperparameters.values import (get_bound_values,
    get_var_name,
    get_var_type,)
var_type = get_var_type(fun_control)
var_name = get_var_name(fun_control)
fun_control.update({"var_type": var_type,
```

```
                    "var_name": var_name})
lower = get_bound_values(fun_control, "lower")
upper = get_bound_values(fun_control, "upper")
```

```
from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control))
```

```
| name         | type   | default   |   lower |   upper | transform             |
|--------------|--------|-----------|---------|---------|-----------------------|
| _L_in        | int    | 10        |    10   |    10   | None                  |
| _L_out       | int    | 1         |     1   |     1   | None                  |
| l1           | int    | 3         |     3   |     8   | transform_power_2_int |
| dropout_prob | float  | 0.01      |     0   |     0.9 | None                  |
| lr_mult      | float  | 1.0       |     0.1 |    10   | None                  |
| batch_size   | int    | 4         |     1   |     4   | transform_power_2_int |
| epochs       | int    | 4         |     2   |    16   | transform_power_2_int |
| k_folds      | int    | 1         |     1   |     1   | None                  |
| patience     | int    | 2         |     3   |     7   | transform_power_2_int |
| optimizer    | factor | SGD       |     0   |     6   | None                  |
| sgd_momentum | float  | 0.0       |     0   |     1   | None                  |
```

19.8.2 The Objective Function `fun_torch`

The objective function `fun_torch` is selected next. It implements an interface from `PyTorch`'s training, validation, and testing methods to `spotPython`.

```
from spotPython.fun.hypertorch import HyperTorch
fun = HyperTorch().fun_torch
```

```
from spotPython.hyperparameters.values import get_default_hyperparameters_as_array
hyper_dict=TorchHyperDict().load()
X_start = get_default_hyperparameters_as_array(fun_control, hyper_dict)
```

19.8.3 Starting the Hyperparameter Tuning

The `spotPython` hyperparameter tuning is started by calling the `Spot` function as described in Section 14.8.4.

```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: 0.1510398238897324 | Loss: 0.0369136862358765 | Epoch: 2 |

MeanAbsoluteError: 0.1342794895172119 | Loss: 0.0284741556418962 | Epoch: 3 |

MeanAbsoluteError: 0.1387000679969788 | Loss: 0.0311537492088974 | Epoch: 4 | MeanAbsoluteErr

MeanAbsoluteError: 0.1153899505734444 | Loss: 0.0220005156741919 | Epoch: 11 | MeanAbsoluteEr
```

```
MeanAbsoluteError: 0.1060830280184746 | Loss: 0.0184431543426686 | Epoch: 14 |

MeanAbsoluteError: 0.1025700047612190 | Loss: 0.0180818413041140 | Epoch: 15 | MeanAbsoluteE

MeanAbsoluteError: 0.0912324488162994 | Loss: 0.0142558935269909 | Epoch: 22 | MeanAbsoluteE

MeanAbsoluteError: 0.0944187492132187 | Loss: 0.0152990247037164 | Epoch: 25 |

MeanAbsoluteError: 0.0998964756727219 | Loss: 0.0166331721978311 | Epoch: 26 | MeanAbsoluteE

MeanAbsoluteError: 0.0826965868473053 | Loss: 0.0118968740173027 | Epoch: 33 | MeanAbsoluteE

MeanAbsoluteError: 0.0851385071873665 | Loss: 0.0124670029372761 | Epoch: 36 |

MeanAbsoluteError: 0.0898290649056435 | Loss: 0.0135611805537912 | Epoch: 37 | MeanAbsoluteE

MeanAbsoluteError: 0.0796431973576546 | Loss: 0.0111558991086081 | Epoch: 44 | MeanAbsoluteE

MeanAbsoluteError: 0.0807356089353561 | Loss: 0.0114866402735443 | Epoch: 47 |

MeanAbsoluteError: 0.0775308609008789 | Loss: 0.0098330018196353 | Epoch: 48 | MeanAbsoluteE

MeanAbsoluteError: 0.0682594180107117 | Loss: 0.0091185896400068 | Epoch: 55 | MeanAbsoluteE

MeanAbsoluteError: 0.0721961930394173 | Loss: 0.0097299493837023 | Epoch: 58 |

MeanAbsoluteError: 0.0750273242592812 | Loss: 0.0108655626383169 | Epoch: 59 | MeanAbsoluteE

MeanAbsoluteError: 0.0618213452398777 | Loss: 0.0073686307633149 | Epoch: 66 | MeanAbsoluteE

MeanAbsoluteError: 0.0683439522981644 | Loss: 0.0088093732881948 | Epoch: 69 |
```

```
MeanAbsoluteError: 0.0711942091584206 | Loss: 0.0089537038216612 | Epoch: 70 | MeanAbsoluteE
Returned to Spot: Validation loss: 0.007897381287827892

config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.19981931523998656, 'lr_mult':
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("Adadelta.__init__() go
Returned to Spot: Validation loss: nan

config: {'_L_in': 10, '_L_out': 1, 'l1': 128, 'dropout_prob': 0.8582565260508446, 'lr_mult':
Error in Net_Core. Call to evaluate_hold_out() failed. err=TypeError("ASGD.__init__() got an
Returned to Spot: Validation loss: nan

config: {'_L_in': 10, '_L_out': 1, 'l1': 16, 'dropout_prob': 0.1773189149831582, 'lr_mult':
Epoch: 1 |


MeanAbsoluteError: 0.1358871459960938 | Loss: 0.0292325268641192 | Epoch: 2 |


MeanAbsoluteError: 0.1287250220775604 | Loss: 0.0263075754170616 | Epoch: 3 |


MeanAbsoluteError: 0.1238213703036308 | Loss: 0.0257791999386003 | Epoch: 4 | MeanAbsoluteEr

config: {'_L_in': 10, '_L_out': 1, 'l1': 32, 'dropout_prob': 0.3840970624671163, 'lr_mult':
Epoch: 1 | MeanAbsoluteError: 0.1656051874160767 | Loss: 0.0439814238465930 | Epoch: 2 | Mea


MeanAbsoluteError: 0.1283811628818512 | Loss: 0.0242146484455780 | Epoch: 7 | MeanAbsoluteEr


MeanAbsoluteError: 0.1149511188268661 | Loss: 0.0216100549472398 | Epoch: 9 | MeanAbsoluteEr


MeanAbsoluteError: 0.1042730063199997 | Loss: 0.0184601759292970 | Epoch: 11 | MeanAbsoluteE


MeanAbsoluteError: 0.0817024111747742 | Loss: 0.0121538409730420 | Epoch: 19 | MeanAbsoluteE


MeanAbsoluteError: 0.0760509595274925 | Loss: 0.0101502324687317 | Epoch: 21 | MeanAbsoluteE


MeanAbsoluteError: 0.0730659738183022 | Loss: 0.0094420600002386 | Epoch: 23 | MeanAbsoluteE


MeanAbsoluteError: 0.0678534358739853 | Loss: 0.0079786121005830 | Epoch: 31 | MeanAbsoluteE


MeanAbsoluteError: 0.0635221004486084 | Loss: 0.0074245966511386 | Epoch: 33 | MeanAbsoluteE
```

MeanAbsoluteError: 0.0604488886892796 | Loss: 0.0070250603450021 | Epoch: 35 | MeanAbsoluteE

MeanAbsoluteError: 0.0537487976253033 | Loss: 0.0057425845175442 | Epoch: 43 | MeanAbsoluteE

MeanAbsoluteError: 0.0632338896393776 | Loss: 0.0066828819874086 | Epoch: 45 | MeanAbsoluteE

MeanAbsoluteError: 0.0548425167798996 | Loss: 0.0058890535346061 | Epoch: 47 | MeanAbsoluteE

MeanAbsoluteError: 0.0673684626817703 | Loss: 0.0082628525591357 | Epoch: 55 | MeanAbsoluteE

MeanAbsoluteError: 0.0626379549503326 | Loss: 0.0066716846470770 | Epoch: 57 | MeanAbsoluteE

MeanAbsoluteError: 0.0611693300306797 | Loss: 0.0068331375338235 | Epoch: 59 | MeanAbsoluteE

MeanAbsoluteError: 0.0579758584499359 | Loss: 0.0053211053539264 | Epoch: 67 | MeanAbsoluteE

MeanAbsoluteError: 0.0569623447954655 | Loss: 0.0058423693423576 | Epoch: 69 | MeanAbsoluteE

MeanAbsoluteError: 0.0619221106171608 | Loss: 0.0074092994572742 | Epoch: 71 | MeanAbsoluteE

MeanAbsoluteError: 0.0643455907702446 | Loss: 0.0073397010748618 | Epoch: 79 | MeanAbsoluteE

MeanAbsoluteError: 0.0700333714485168 | Loss: 0.0078033572307935 | Epoch: 81 | MeanAbsoluteE

MeanAbsoluteError: 0.0538323856890202 | Loss: 0.0059160708869489 | Epoch: 83 | MeanAbsoluteE

MeanAbsoluteError: 0.0657727196812630 | Loss: 0.0083600583755852 | Epoch: 91 | MeanAbsoluteE

MeanAbsoluteError: 0.0668385848402977 | Loss: 0.0073925533552507 | Epoch: 93 | MeanAbsoluteE

MeanAbsoluteError: 0.0540138110518456 | Loss: 0.0058280026423745 | Epoch: 95 | MeanAbsoluteE

MeanAbsoluteError: 0.0500309988856316 | Loss: 0.0046575507363550 | Epoch: 103 | MeanAbsolute

MeanAbsoluteError: 0.0564113855361938 | Loss: 0.0066059492467167 | Epoch: 105 | MeanAbsolute

MeanAbsoluteError: 0.0570416375994682 | Loss: 0.0058915256859588 | Epoch: 107 | MeanAbsolute

MeanAbsoluteError: 0.0640984252095222 | Loss: 0.0073109539899681 | Epoch: 115 | MeanAbsoluteE

MeanAbsoluteError: 0.0610482282936573 | Loss: 0.0073482567646639 | Epoch: 117 | MeanAbsoluteE

MeanAbsoluteError: 0.0628093183040619 | Loss: 0.0074057867995610 | Epoch: 119 | MeanAbsoluteE

MeanAbsoluteError: 0.0467109382152557 | Loss: 0.0048273830746582 | Epoch: 127 | MeanAbsoluteE

MeanAbsoluteError: 0.0494071915745735 | Loss: 0.0052260657294506 | Epoch: 129 | MeanAbsoluteE

MeanAbsoluteError: 0.0527051575481892 | Loss: 0.0046347018827586 | Epoch: 131 | MeanAbsoluteE

MeanAbsoluteError: 0.0679387673735619 | Loss: 0.0075022486914685 | Epoch: 139 | MeanAbsoluteE

MeanAbsoluteError: 0.0543936081230640 | Loss: 0.0063156054606416 | Epoch: 141 | MeanAbsoluteE

MeanAbsoluteError: 0.0618858784437180 | Loss: 0.0088248628522515 | Epoch: 143 | MeanAbsoluteE

MeanAbsoluteError: 0.0478628464043140 | Loss: 0.0050847133710417 | Epoch: 151 | MeanAbsoluteE

MeanAbsoluteError: 0.0595661662518978 | Loss: 0.0062492241911394 | Epoch: 153 | MeanAbsoluteE

MeanAbsoluteError: 0.0559690110385418 | Loss: 0.0059656762720184 | Epoch: 155 | MeanAbsoluteE

MeanAbsoluteError: 0.0548455938696861 | Loss: 0.0057135789354920 | Epoch: 163 | MeanAbsoluteE

MeanAbsoluteError: 0.0505605116486549 | Loss: 0.0047682228246949 | Epoch: 165 | MeanAbsoluteE

MeanAbsoluteError: 0.0672666281461716 | Loss: 0.0077738489612545 | Epoch: 167 | MeanAbsoluteE

MeanAbsoluteError: 0.0483722575008869 | Loss: 0.0050010307114510 | Epoch: 175 | MeanAbsoluteE

Epoch: 176 | MeanAbsoluteError: 0.0525178872048855 | Loss: 0.0054417301833286 | Epoch: 177 |

MeanAbsoluteError: 0.0640534311532974 | Loss: 0.0069467917476830 | Epoch: 178 | MeanAbsoluteE

MeanAbsoluteError: 0.0503111854195595 | Loss: 0.0057946335230219 |

```
Epoch: 187 | MeanAbsoluteError: 0.0563249513506889 | Loss: 0.0059949087567235 | Epoch: 188 |

MeanAbsoluteError: 0.0544509626924992 | Loss: 0.0056156209014405 | Epoch: 190 | MeanAbsoluteE

MeanAbsoluteError: 0.0529893450438976 | Loss: 0.0064462734367944 | Epoch: 199 | MeanAbsoluteE

MeanAbsoluteError: 0.0444929152727127 | Loss: 0.0038233834983609 | Epoch: 202 | MeanAbsoluteE

MeanAbsoluteError: 0.0431372486054897 | Loss: 0.0041864182827627 | Epoch: 211 | MeanAbsoluteE

MeanAbsoluteError: 0.0526904948055744 | Loss: 0.0055840924523134 | Epoch: 214 | MeanAbsoluteE

MeanAbsoluteError: 0.0507537424564362 | Loss: 0.0059442198038787 | Epoch: 223 | MeanAbsoluteE

MeanAbsoluteError: 0.0548335276544094 | Loss: 0.0061971954633727 | Epoch: 226 | MeanAbsoluteE

MeanAbsoluteError: 0.0435824096202850 | Loss: 0.0040297141102584 | Epoch: 235 | MeanAbsoluteE

MeanAbsoluteError: 0.0555229634046555 | Loss: 0.0056177166958438 | Epoch: 238 | MeanAbsoluteE

MeanAbsoluteError: 0.0495964139699936 | Loss: 0.0057975444896759 | Epoch: 247 | MeanAbsoluteE

MeanAbsoluteError: 0.0459261685609818 | Loss: 0.0049366121713415 | Epoch: 250 | MeanAbsoluteE


config: {'_L_in': 10, '_L_out': 1, 'l1': 8, 'dropout_prob': 0.01403604591958968, 'lr_mult': 6
Epoch: 1 | MeanAbsoluteError: 0.1259300410747528 | Loss: 0.0265161717237022 | Epoch: 2 | Mean

MeanAbsoluteError: 0.0327887088060379 | Loss: 0.0035031061339204 | Epoch: 13 | MeanAbsoluteEr

MeanAbsoluteError: 0.0164080802351236 | Loss: 0.0018185865623022 | Epoch: 25 | MeanAbsoluteEr

MeanAbsoluteError: 0.0305399838835001 | Loss: 0.0025158639749095 | Epoch: 37 | MeanAbsoluteEr

MeanAbsoluteError: 0.0255678538233042 | Loss: 0.0014882139156199 | Epoch: 49 | MeanAbsoluteEr

MeanAbsoluteError: 0.0120021598413587 | Loss: 0.0005452528115424 | Epoch: 61 | MeanAbsoluteEr
```

```
MeanAbsoluteError: 0.0209484621882439 | Loss: 0.0012815237752971 | Epoch: 73 | MeanAbsoluteE
MeanAbsoluteError: 0.0244137868285179 | Loss: 0.0012053348450297 | Epoch: 85 | MeanAbsoluteE
MeanAbsoluteError: 0.0143645545467734 | Loss: 0.0006391305000043 | Epoch: 97 | MeanAbsoluteE
MeanAbsoluteError: 0.0144806001335382 | Loss: 0.0003149038960768 | Epoch: 109 | MeanAbsolute
MeanAbsoluteError: 0.0194163508713245 | Loss: 0.0007067082903967 | Epoch: 121 | MeanAbsolute
MeanAbsoluteError: 0.0204860307276249 | Loss: 0.0006097237875158 | Epoch: 133 | MeanAbsolute
MeanAbsoluteError: 0.0144913559779525 | Loss: 0.0011939185611051 | Epoch: 145 | MeanAbsolute
MeanAbsoluteError: 0.0214977841824293 | Loss: 0.0012670696512367 | Epoch: 157 | MeanAbsolute
MeanAbsoluteError: 0.0160639379173517 | Loss: 0.0010505288375594 | Epoch: 169 | MeanAbsolute
MeanAbsoluteError: 0.0077423905022442 | Loss: 0.0003760929630995 | Epoch: 181 | MeanAbsolute
MeanAbsoluteError: 0.0106047233566642 | Loss: 0.0003640439988789 | Epoch: 193 | MeanAbsolute
MeanAbsoluteError: 0.0105459010228515 | Loss: 0.0006691445595559 | Epoch: 205 | MeanAbsolute
MeanAbsoluteError: 0.0064728395082057 | Loss: 0.0004901313764692 | Epoch: 217 | MeanAbsolute
MeanAbsoluteError: 0.0088179251179099 | Loss: 0.0002607876748171 | Epoch: 229 | MeanAbsolute
MeanAbsoluteError: 0.0124101229012012 | Loss: 0.0017336170468824 | Epoch: 241 | MeanAbsolute
MeanAbsoluteError: 0.0078885918483138 | Loss: 0.0005159430379296 | Epoch: 253 | MeanAbsolute
MeanAbsoluteError: 0.0099451206624508 | Loss: 0.0005569837398741 | Epoch: 265 | MeanAbsolute
MeanAbsoluteError: 0.0047567230649292 | Loss: 0.0000650893727509 | Epoch: 277 | MeanAbsolute
MeanAbsoluteError: 0.0071909665130079 | Loss: 0.0002218923047239 | Epoch: 289 | MeanAbsolute
```

```
MeanAbsoluteError: 0.0073184841312468 | Loss: 0.0003039859170473 | Epoch: 301 | MeanAbsoluteE

MeanAbsoluteError: 0.0052791298367083 | Loss: 0.0004098871385736 | Epoch: 313 | MeanAbsoluteE

MeanAbsoluteError: 0.0145234661176801 | Loss: 0.0011255679972620 | Epoch: 325 | MeanAbsoluteE

MeanAbsoluteError: 0.0257583260536194 | Loss: 0.0012750036694342 | Epoch: 337 | MeanAbsoluteE

MeanAbsoluteError: 0.0128293428570032 | Loss: 0.0004059095747499 | Epoch: 349 | MeanAbsoluteE

MeanAbsoluteError: 0.0077029885724187 | Loss: 0.0001330500907371 | Epoch: 361 | MeanAbsoluteE

MeanAbsoluteError: 0.0055201915092766 | Loss: 0.0001149734919540 | Epoch: 373 | MeanAbsoluteE

MeanAbsoluteError: 0.0107059990987182 | Loss: 0.0003656530180754 | Epoch: 385 | MeanAbsoluteE

MeanAbsoluteError: 0.0051011946052313 | Loss: 0.0002807884380242 | Epoch: 397 | MeanAbsoluteE

MeanAbsoluteError: 0.0065328436903656 | Loss: 0.0002359646889126 | Epoch: 409 | MeanAbsoluteE

MeanAbsoluteError: 0.0130674354732037 | Loss: 0.0002542652739641 | Epoch: 421 | MeanAbsoluteE

MeanAbsoluteError: 0.0074885580688715 | Loss: 0.0001743301079985 | Epoch: 433 | MeanAbsoluteE
Returned to Spot: Validation loss: 0.0012166352798885782

spotPython tuning: 0.0012166352798885782 [#---------] 13.79%


config: {'_L_in': 10, '_L_out': 1, 'l1': 8, 'dropout_prob': 0.007840397692637325, 'lr_mult':
Epoch: 1 | MeanAbsoluteError: 0.1205963119864464 | Loss: 0.0241384317877849 | Epoch: 2 | Mean

MeanAbsoluteError: 0.0169818513095379 | Loss: 0.0017830141591012 | Epoch: 13 | MeanAbsoluteEr

MeanAbsoluteError: 0.0125633608549833 | Loss: 0.0009501198876119 | Epoch: 25 | MeanAbsoluteEr

MeanAbsoluteError: 0.0118321692571044 | Loss: 0.0027884146868653 | Epoch: 37 | MeanAbsoluteEr
```

```
MeanAbsoluteError: 0.0141557808965445 | Loss: 0.0005133587056710 | Epoch: 49 | MeanAbsoluteE

MeanAbsoluteError: 0.0088058169931173 | Loss: 0.0002694251309214 | Epoch: 61 | MeanAbsoluteE

MeanAbsoluteError: 0.0098094344139099 | Loss: 0.0013321367274138 | Epoch: 73 | MeanAbsoluteE

MeanAbsoluteError: 0.0132227297872305 | Loss: 0.0005798433554255 | Epoch: 85 | MeanAbsoluteE

MeanAbsoluteError: 0.0151141136884689 | Loss: 0.0010607976280306 | Epoch: 97 | MeanAbsoluteE

MeanAbsoluteError: 0.0176987703889608 | Loss: 0.0008667116329123 | Epoch: 109 | MeanAbsolute

MeanAbsoluteError: 0.0083112176507711 | Loss: 0.0001808526781372 | Epoch: 121 | MeanAbsolute

MeanAbsoluteError: 0.0150752114132047 | Loss: 0.0007542517166592 | Epoch: 133 | MeanAbsolute

MeanAbsoluteError: 0.0135327577590942 | Loss: 0.0006407793026531 | Epoch: 145 | MeanAbsolute

MeanAbsoluteError: 0.0116811478510499 | Loss: 0.0007830665716890 | Epoch: 157 | MeanAbsolute

MeanAbsoluteError: 0.0171964224427938 | Loss: 0.0008979361139753 | Epoch: 169 | MeanAbsolute

MeanAbsoluteError: 0.0211204513907433 | Loss: 0.0012211554498399 | Epoch: 181 | MeanAbsolute

MeanAbsoluteError: 0.0112252272665501 | Loss: 0.0008959564197061 | Epoch: 193 | MeanAbsolute

MeanAbsoluteError: 0.0048602335155010 | Loss: 0.0001829798261257 | Epoch: 205 | MeanAbsolute

MeanAbsoluteError: 0.0165717359632254 | Loss: 0.0008664759170164 | Epoch: 217 | MeanAbsolute

MeanAbsoluteError: 0.0084485383704305 | Loss: 0.0002514461468644 | Epoch: 229 | MeanAbsolute

MeanAbsoluteError: 0.0080088237300515 | Loss: 0.0003094237603675 | Epoch: 241 | MeanAbsolute

MeanAbsoluteError: 0.0108943609520793 | Loss: 0.0006645920369632 | Epoch: 253 | MeanAbsolute

MeanAbsoluteError: 0.0113977165892720 | Loss: 0.0009510856697919 | Epoch: 265 | MeanAbsolute
```

```
MeanAbsoluteError: 0.0128106558695436 | Loss: 0.0005864251575794 | Epoch: 277 | MeanAbsoluteE

MeanAbsoluteError: 0.0138528663665056 | Loss: 0.0003623745913884 | Epoch: 289 | MeanAbsoluteE

MeanAbsoluteError: 0.0071873678825796 | Loss: 0.0002182312309024 | Epoch: 301 | MeanAbsoluteE

MeanAbsoluteError: 0.0173418503254652 | Loss: 0.0006440675026265 | Epoch: 313 | MeanAbsoluteE

MeanAbsoluteError: 0.0122947813943028 | Loss: 0.0003349522400975 | Epoch: 325 | MeanAbsoluteE

MeanAbsoluteError: 0.0094737457111478 | Loss: 0.0005221923499305 | Epoch: 337 | MeanAbsoluteE
Returned to Spot: Validation loss: 0.0006423751242077982

spotPython tuning: 0.0006423751242077982 [###-------] 25.18%


config: {'_L_in': 10, '_L_out': 1, 'l1': 8, 'dropout_prob': 0.7079001982078938, 'lr_mult': 9
Epoch: 1 | MeanAbsoluteError: 0.1394856721162796 | Loss: 0.0312266622326876 | Epoch: 2 | Mean
Returned to Spot: Validation loss: 0.032126733248955326

spotPython tuning: 0.0006423751242077982 [###-------] 26.46%


config: {'_L_in': 10, '_L_out': 1, 'l1': 8, 'dropout_prob': 0.0, 'lr_mult': 10.0, 'batch_size
Epoch: 1 | MeanAbsoluteError: 0.1113698855042458 | Loss: 0.0198563025087902 | Epoch: 2 | Mean

MeanAbsoluteError: 0.0007408881210722 | Loss: 0.0000008221615172 | Epoch: 13 | MeanAbsoluteEr

MeanAbsoluteError: 0.0010000267066061 | Loss: 0.0000015226560377 | Epoch: 26 | MeanAbsoluteEr

MeanAbsoluteError: 0.0022836062125862 | Loss: 0.0000061866831246 | Epoch: 39 | MeanAbsoluteEr

MeanAbsoluteError: 0.0022473083809018 | Loss: 0.0000071235882465 | Epoch: 51 | MeanAbsoluteEr

MeanAbsoluteError: 0.0012773519847542 | Loss: 0.0000026701988105 | Epoch: 64 | MeanAbsoluteEr

MeanAbsoluteError: 0.0027892016805708 | Loss: 0.0000093172711188 | Epoch: 76 | MeanAbsoluteEr
```

```
MeanAbsoluteError: 0.0044077322818339 | Loss: 0.0000226608486676 | Epoch: 89 | MeanAbsoluteE

MeanAbsoluteError: 0.0081684673205018 | Loss: 0.0000695307639705 | Epoch: 101 | MeanAbsolute

MeanAbsoluteError: 0.0071641979739070 | Loss: 0.0000532125335380 | Epoch: 114 | MeanAbsolute

MeanAbsoluteError: 0.0077573400922120 | Loss: 0.0000625956732907 | Epoch: 127 | MeanAbsolute

MeanAbsoluteError: 0.0013754771789536 | Loss: 0.0000026736984885 | Epoch: 140 | MeanAbsolute
Returned to Spot: Validation loss: 0.0001167130632031905

spotPython tuning: 0.0001167130632031905 [###-------] 32.14%


config: {'_L_in': 10, '_L_out': 1, 'l1': 64, 'dropout_prob': 0.7103122103038131, 'lr_mult': 3
Epoch: 1 | MeanAbsoluteError: 0.1629366874694824 | Loss: 0.0404722033509691 | Epoch: 2 | Mean

MeanAbsoluteError: 0.1238151937723160 | Loss: 0.0236612408189103 | Epoch: 12 | MeanAbsoluteE

MeanAbsoluteError: 0.1004069894552231 | Loss: 0.0171346596050027 | Epoch: 23 | MeanAbsoluteE

MeanAbsoluteError: 0.0887333527207375 | Loss: 0.0128588269244095 | Epoch: 34 | MeanAbsoluteE

MeanAbsoluteError: 0.0924901738762856 | Loss: 0.0144079264290141 | Epoch: 45 | MeanAbsoluteE
Returned to Spot: Validation loss: 0.012755457685623122

spotPython tuning: 0.0001167130632031905 [###-------] 34.72%


config: {'_L_in': 10, '_L_out': 1, 'l1': 8, 'dropout_prob': 0.01047915105991038, 'lr_mult': 1
Epoch: 1 | MeanAbsoluteError: 0.1303636133670807 | Loss: 0.0290535452138436 | Epoch: 2 | Mean

MeanAbsoluteError: 0.0448841191828251 | Loss: 0.0039945771124238 | Epoch: 13 | MeanAbsoluteE

MeanAbsoluteError: 0.0202892404049635 | Loss: 0.0032613144183798 | Epoch: 25 | MeanAbsoluteE

MeanAbsoluteError: 0.0124744810163975 | Loss: 0.0002864225737683 | Epoch: 37 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.0306338779628277 | Loss: 0.0021599373844526 | Epoch: 49 | MeanAbsoluteE

MeanAbsoluteError: 0.0096576791256666 | Loss: 0.0011781135661797 | Epoch: 61 | MeanAbsoluteE

MeanAbsoluteError: 0.0109535129740834 | Loss: 0.0017360518089079 | Epoch: 73 | MeanAbsoluteE

MeanAbsoluteError: 0.0104894079267979 | Loss: 0.0007397842546197 | Epoch: 85 | MeanAbsoluteE

MeanAbsoluteError: 0.0073121627792716 | Loss: 0.0006475056880983 | Epoch: 97 | MeanAbsoluteE

MeanAbsoluteError: 0.0193673968315125 | Loss: 0.0013365803893784 | Epoch: 109 | MeanAbsolute

MeanAbsoluteError: 0.0152108799666166 | Loss: 0.0012990944780636 | Epoch: 121 | MeanAbsolute

MeanAbsoluteError: 0.0206849221140146 | Loss: 0.0015691996028947 | Epoch: 133 | MeanAbsolute

MeanAbsoluteError: 0.0094573497772217 | Loss: 0.0012092308610602 | Epoch: 145 | MeanAbsolute

MeanAbsoluteError: 0.0109200337901711 | Loss: 0.0017667702018954 | Epoch: 157 | MeanAbsolute

MeanAbsoluteError: 0.0142289400100708 | Loss: 0.0029116271556644 | Epoch: 169 | MeanAbsolute

MeanAbsoluteError: 0.0197799075394869 | Loss: 0.0019894990844860 | Epoch: 181 | MeanAbsolute

MeanAbsoluteError: 0.0147574348375201 | Loss: 0.0030820557047647 | Epoch: 193 | MeanAbsolute

MeanAbsoluteError: 0.0170524008572102 | Loss: 0.0012645274963640 | Epoch: 205 | MeanAbsolute

MeanAbsoluteError: 0.0140438172966242 | Loss: 0.0012942912664202 | Epoch: 217 | MeanAbsolute

MeanAbsoluteError: 0.0102257439866662 | Loss: 0.0009690678493586 | Epoch: 229 | MeanAbsolute
Returned to Spot: Validation loss: 0.00218908519405216

spotPython tuning: 0.0001167130632031905 [####------] 42.66%


config: {'_L_in': 10, '_L_out': 1, 'l1': 8, 'dropout_prob': 0.0, 'lr_mult': 0.1, 'batch_size
Epoch: 1 | MeanAbsoluteError: 0.1666615754365921 | Loss: 0.0403395072115879 | Epoch: 2 | Mea
```

```
spotPython tuning: 0.0001167130632031905 [####------] 43.71%


config: {'_L_in': 10, '_L_out': 1, 'l1': 8, 'dropout_prob': 0.0, 'lr_mult': 0.10225915259607
Epoch: 1 | MeanAbsoluteError: 0.4935512840747833 | Loss: 0.2715650879238781 | Epoch: 2 | Mea

spotPython tuning: 0.0001167130632031905 [#####-----] 45.95%


config: {'_L_in': 10, '_L_out': 1, 'l1': 16, 'dropout_prob': 0.09207636896183456, 'lr_mult':
Epoch: 1 | MeanAbsoluteError: 0.6832665205001831 | Loss: 0.4971198408227218 | Epoch: 2 | Mea

MeanAbsoluteError: 0.1315160840749741 | Loss: 0.0312749911344757 | Epoch: 13 | MeanAbsoluteE

spotPython tuning: 0.0001167130632031905 [#####-----] 47.59%


config: {'_L_in': 10, '_L_out': 1, 'l1': 64, 'dropout_prob': 0.42282553454816796, 'lr_mult':
Epoch: 1 | MeanAbsoluteError: 0.1557978987693787 | Loss: 0.0366417783998737 | Epoch: 2 | Mea

MeanAbsoluteError: 0.0927854180335999 | Loss: 0.0150447305794315 | Epoch: 12 | MeanAbsoluteE

MeanAbsoluteError: 0.0611935928463936 | Loss: 0.0068885862190080 | Epoch: 23 | MeanAbsoluteE

MeanAbsoluteError: 0.0535407401621342 | Loss: 0.0048315608935235 | Epoch: 34 | MeanAbsoluteE

MeanAbsoluteError: 0.0505434572696686 | Loss: 0.0058777173690032 | Epoch: 45 | MeanAbsoluteE

Epoch: 56 | MeanAbsoluteError: 0.0507094971835613 | Loss: 0.0051756534866351 | Epoch: 57 | M

MeanAbsoluteError: 0.0559190288186073 | Loss: 0.0064316885913477 | Epoch: 67 | MeanAbsoluteE

MeanAbsoluteError: 0.0559693276882172 | Loss: 0.0061769057187791 | Epoch: 78 | MeanAbsoluteE

MeanAbsoluteError: 0.0527518726885319 | Loss: 0.0053266166222257 | Epoch: 89 | MeanAbsoluteE

MeanAbsoluteError: 0.0511462576687336 | Loss: 0.0050397202063131 | Epoch: 100 | MeanAbsolute
```

```
MeanAbsoluteError: 0.0573087930679321 | Loss: 0.0064295307720864 | Epoch: 111 | MeanAbsoluteE

MeanAbsoluteError: 0.0459469445049763 | Loss: 0.0041253883973695 | Epoch: 122 | MeanAbsoluteE
Returned to Spot: Validation loss: 0.005531792463060762

spotPython tuning: 0.0001167130632031905 [#####-----] 52.68%


config: {'_L_in': 10, '_L_out': 1, 'l1': 8, 'dropout_prob': 0.047310570678932046, 'lr_mult':
Epoch: 1 | MeanAbsoluteError: 0.1269474327564240 | Loss: 0.0258942560819713 | Epoch: 2 | Mean

MeanAbsoluteError: 0.1269347816705704 | Loss: 0.0259025318266236 | Epoch: 13 | MeanAbsoluteEr

MeanAbsoluteError: 0.1269347071647644 | Loss: 0.0259025015770213 | Epoch: 25 | MeanAbsoluteEr
Returned to Spot: Validation loss: 0.02590249361482596

spotPython tuning: 0.0001167130632031905 [#####-----] 54.74%


config: {'_L_in': 10, '_L_out': 1, 'l1': 8, 'dropout_prob': 0.0, 'lr_mult': 5.43459415534586
Epoch: 1 | MeanAbsoluteError: 0.1216830834746361 | Loss: 0.0228505738716769 | Epoch: 2 | Mean

MeanAbsoluteError: 0.0455204583704472 | Loss: 0.0032334629918056 | Epoch: 14 | MeanAbsoluteEr

MeanAbsoluteError: 0.0034283320419490 | Loss: 0.0000183603071707 | Epoch: 27 | MeanAbsoluteEr

MeanAbsoluteError: 0.0007078434573486 | Loss: 0.0000007885536924 | Epoch: 40 | MeanAbsoluteEr

MeanAbsoluteError: 0.0007937217014842 | Loss: 0.0000009786500607 | Epoch: 53 | MeanAbsoluteEr

MeanAbsoluteError: 0.0014183936873451 | Loss: 0.0000026869568525 | Epoch: 66 | MeanAbsoluteEr

MeanAbsoluteError: 0.0007167390431277 | Loss: 0.0000008083861608 | Epoch: 79 | MeanAbsoluteEr

MeanAbsoluteError: 0.0021095594856888 | Loss: 0.0000053434030059 | Epoch: 92 | MeanAbsoluteEr

MeanAbsoluteError: 0.0011429756414145 | Loss: 0.0000019536805734 | Epoch: 105 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.0030608482193202 | Loss: 0.0000103568131755 | Epoch: 118 | MeanAbsoluteE

MeanAbsoluteError: 0.0026806255336851 | Loss: 0.0000081586421474 | Epoch: 131 | MeanAbsoluteE

MeanAbsoluteError: 0.0026817990001291 | Loss: 0.0000081562308854 | Epoch: 144 | MeanAbsoluteE

MeanAbsoluteError: 0.0025949000846595 | Loss: 0.0000076917069512 | Epoch: 157 | MeanAbsoluteE

MeanAbsoluteError: 0.0026896544732153 | Loss: 0.0000081921099608 | Epoch: 170 | MeanAbsoluteE

MeanAbsoluteError: 0.0024968709331006 | Loss: 0.0000071885608098 | Epoch: 183 | MeanAbsoluteE

MeanAbsoluteError: 0.0026650638319552 | Loss: 0.0000080562588209 | Epoch: 196 | MeanAbsoluteE
Returned to Spot: Validation loss: 8.020244889083092e-06

spotPython tuning: 8.020244889083092e-06 [######----] 61.66%


config: {'_L_in': 10, '_L_out': 1, 'l1': 8, 'dropout_prob': 0.3116249342762122, 'lr_mult': 5
Epoch: 1 | MeanAbsoluteError: 0.2691048383712769 | Loss: 0.0961951687932014 | Epoch: 2 | Mean

spotPython tuning: 8.020244889083092e-06 [######----] 63.20%


config: {'_L_in': 10, '_L_out': 1, 'l1': 8, 'dropout_prob': 0.0, 'lr_mult': 6.21943619092621
Epoch: 1 | MeanAbsoluteError: 0.1004863902926445 | Loss: 0.0160443981447698 | Epoch: 2 | Mean

MeanAbsoluteError: 0.0302014295011759 | Loss: 0.0014244260206684 | Epoch: 14 | MeanAbsoluteEr

MeanAbsoluteError: 0.0008602430461906 | Loss: 0.0000011578253827 | Epoch: 27 | MeanAbsoluteEr

MeanAbsoluteError: 0.0008761439821683 | Loss: 0.0000011696610696 | Epoch: 40 | MeanAbsoluteEr

MeanAbsoluteError: 0.0008276483858936 | Loss: 0.0000010432452761 | Epoch: 53 | MeanAbsoluteEr

MeanAbsoluteError: 0.0011258029844612 | Loss: 0.0000018900483502 | Epoch: 66 | MeanAbsoluteEr

MeanAbsoluteError: 0.0008460177923553 | Loss: 0.0000010771375214 | Epoch: 79 | MeanAbsoluteEr
```

```
MeanAbsoluteError: 0.0023655928671360 | Loss: 0.0000069890701656 | Epoch: 92 | MeanAbsoluteE

MeanAbsoluteError: 0.0025000469759107 | Loss: 0.0000077429366755 | Epoch: 105 | MeanAbsolute

MeanAbsoluteError: 0.0025418908335268 | Loss: 0.0000079969828388 | Epoch: 118 | MeanAbsolute

MeanAbsoluteError: 0.0025482440833002 | Loss: 0.0000080452463599 | Epoch: 131 | MeanAbsolute

MeanAbsoluteError: 0.0025386775378138 | Loss: 0.0000079989238561 | Epoch: 144 | MeanAbsolute

MeanAbsoluteError: 0.0025224983692169 | Loss: 0.0000079130009597 | Epoch: 157 | MeanAbsolute
Returned to Spot: Validation loss: 7.86182389626782e-06

spotPython tuning: 7.86182389626782e-06 [#######---] 68.97%


config: {'_L_in': 10, '_L_out': 1, 'l1': 8, 'dropout_prob': 0.0, 'lr_mult': 4.742898547325524
Epoch: 1 | MeanAbsoluteError: 0.2303449511528015 | Loss: 0.0737619445120034 | Epoch: 2 | Mean

MeanAbsoluteError: 0.0681812912225723 | Loss: 0.0073025822026753 | Epoch: 24 | MeanAbsoluteE

MeanAbsoluteError: 0.0012985047651455 | Loss: 0.0000023177772583 | Epoch: 47 | MeanAbsoluteE

MeanAbsoluteError: 0.0019470422994345 | Loss: 0.0000046265069949 | Epoch: 70 | MeanAbsoluteE

MeanAbsoluteError: 0.0009877602569759 | Loss: 0.0000015339009445 | Epoch: 93 | MeanAbsoluteE

MeanAbsoluteError: 0.0018487668130547 | Loss: 0.0000042020768122 | Epoch: 116 | MeanAbsolute

MeanAbsoluteError: 0.0023953604977578 | Loss: 0.0000065628210235 | Epoch: 139 | MeanAbsolute

MeanAbsoluteError: 0.0029629098717123 | Loss: 0.0000096644987121 | Epoch: 161 | MeanAbsolute

MeanAbsoluteError: 0.0033621899783611 | Loss: 0.0000122425132615 | Epoch: 184 | MeanAbsolute

MeanAbsoluteError: 0.0036270124837756 | Loss: 0.0000141382268722 | Epoch: 207 | MeanAbsolute
```

```
MeanAbsoluteError: 0.0037186646368355 | Loss: 0.0000148396886384 | Epoch: 230 | MeanAbsoluteE

MeanAbsoluteError: 0.0007760192966089 | Loss: 0.0000009155401654 | Epoch: 252 | MeanAbsoluteE
Returned to Spot: Validation loss: 3.3469929719917434e-06

spotPython tuning: 3.3469929719917434e-06 [########--] 76.08%


config: {'_L_in': 10, '_L_out': 1, 'l1': 8, 'dropout_prob': 0.0, 'lr_mult': 10.0, 'batch_siz
Epoch: 1 | MeanAbsoluteError: 0.1161584332585335 | Loss: 0.0214986904573284 | Epoch: 2 | Mea

MeanAbsoluteError: 0.0011773760197684 | Loss: 0.0000097427915732 | Epoch: 14 | MeanAbsoluteE

MeanAbsoluteError: 0.0012071259552613 | Loss: 0.0000066665472054 | Epoch: 27 | MeanAbsoluteE

MeanAbsoluteError: 0.0022320444695652 | Loss: 0.0000099621252885 | Epoch: 40 | MeanAbsoluteE

MeanAbsoluteError: 0.0009291577152908 | Loss: 0.0000058048990133 | Epoch: 53 | MeanAbsoluteE

MeanAbsoluteError: 0.0039662462659180 | Loss: 0.0000214834309478 | Epoch: 66 | MeanAbsoluteE

MeanAbsoluteError: 0.0077602951787412 | Loss: 0.0000648323562736 | Epoch: 79 | MeanAbsoluteE

MeanAbsoluteError: 0.0077003170736134 | Loss: 0.0000634179732994 | Epoch: 92 | MeanAbsoluteE

MeanAbsoluteError: 0.0078134294599295 | Loss: 0.0000648372841074 | Epoch: 105 | MeanAbsoluteE

MeanAbsoluteError: 0.0078026326373219 | Loss: 0.0000644469567449 | Epoch: 118 | MeanAbsoluteE

MeanAbsoluteError: 0.0077650635503232 | Loss: 0.0000637157023256 | Epoch: 131 | MeanAbsoluteE

MeanAbsoluteError: 0.0077335992828012 | Loss: 0.0000631223633957 | Epoch: 144 | MeanAbsoluteE

MeanAbsoluteError: 0.0076987743377686 | Loss: 0.0000625036609563 | Epoch: 157 | MeanAbsoluteE

MeanAbsoluteError: 0.0076707499101758 | Loss: 0.0000620038621204 | Epoch: 170 | MeanAbsoluteE
Returned to Spot: Validation loss: 6.159489940297422e-05
```

```
spotPython tuning: 3.3469929719917434e-06 [#########-] 86.01%

config: {'_L_in': 10, '_L_out': 1, 'l1': 8, 'dropout_prob': 0.0, 'lr_mult': 6.44438193951320
Epoch: 1 | MeanAbsoluteError: 0.1059617176651955 | Loss: 0.0170538873722156 | Epoch: 2 | Mea

MeanAbsoluteError: 0.0015867420006543 | Loss: 0.0000431853119191 | Epoch: 8 | MeanAbsoluteEr

MeanAbsoluteError: 0.0018079578876495 | Loss: 0.0000066154641680 | Epoch: 15 | MeanAbsoluteE

MeanAbsoluteError: 0.0025978868361562 | Loss: 0.0000082472284854 | Epoch: 22 | MeanAbsoluteE

MeanAbsoluteError: 0.0026154634542763 | Loss: 0.0000093112806098 | Epoch: 29 | MeanAbsoluteE

MeanAbsoluteError: 0.0044224872253835 | Loss: 0.0000221498167472 | Epoch: 36 | MeanAbsoluteE

MeanAbsoluteError: 0.0018610574770719 | Loss: 0.0000052990743461 | Epoch: 43 | MeanAbsoluteE

MeanAbsoluteError: 0.0040572881698608 | Loss: 0.0000180560685112 | Epoch: 50 | MeanAbsoluteE

MeanAbsoluteError: 0.0017881356179714 | Loss: 0.0000048668695878 | Epoch: 57 | MeanAbsoluteE

MeanAbsoluteError: 0.0048836744390428 | Loss: 0.0000256791884021 | Epoch: 64 | MeanAbsoluteE

MeanAbsoluteError: 0.0034400015138090 | Loss: 0.0000131554683685 | Epoch: 71 | MeanAbsoluteE

MeanAbsoluteError: 0.0018266403349116 | Loss: 0.0000048689317335 | Epoch: 78 | MeanAbsoluteE

MeanAbsoluteError: 0.0047149835154414 | Loss: 0.0000239193749197 | Epoch: 85 | MeanAbsoluteE

MeanAbsoluteError: 0.0017824181122705 | Loss: 0.0000044860326303 | Epoch: 92 | MeanAbsoluteE

MeanAbsoluteError: 0.0046313782222569 | Loss: 0.0000230390805518 | Epoch: 99 | MeanAbsoluteE

MeanAbsoluteError: 0.0037343322765082 | Loss: 0.0000153465753950 | Epoch: 106 | MeanAbsolute

MeanAbsoluteError: 0.0015600194456056 | Loss: 0.0000033558154496 | Epoch: 113 | MeanAbsolute
```

```
MeanAbsoluteError: 0.0051116067916155 | Loss: 0.0000277802722364 | Epoch: 120 | MeanAbsoluteE

MeanAbsoluteError: 0.0025119229685515 | Loss: 0.0000078779259790 | Epoch: 127 | MeanAbsoluteE

MeanAbsoluteError: 0.0017276360886171 | Loss: 0.0000041671688700 | Epoch: 134 | MeanAbsoluteE

MeanAbsoluteError: 0.0043615684844553 | Loss: 0.0000205602725206 | Epoch: 141 | MeanAbsoluteE

MeanAbsoluteError: 0.0012670627329499 | Loss: 0.0000025011126381 | Epoch: 148 | MeanAbsoluteE
Returned to Spot: Validation loss: 2.1475712613513073e-05

spotPython tuning: 3.3469929719917434e-06 [##########] 99.17%


config: {'_L_in': 10, '_L_out': 1, 'l1': 64, 'dropout_prob': 0.026440387900600198, 'lr_mult'
Epoch: 1 | MeanAbsoluteError: 0.1200591623783112 | Loss: 0.0226752350204869 | Epoch: 2 | Mean

MeanAbsoluteError: 0.0265268385410309 | Loss: 0.0013212607407599 | Epoch: 20 | MeanAbsoluteEr

MeanAbsoluteError: 0.0262160785496235 | Loss: 0.0009835676051137 | Epoch: 39 | MeanAbsoluteEr

MeanAbsoluteError: 0.0249809212982655 | Loss: 0.0012966261649700 | Epoch: 58 | MeanAbsoluteEr

MeanAbsoluteError: 0.0191431678831577 | Loss: 0.0006439807077282 | Epoch: 77 | MeanAbsoluteEr

MeanAbsoluteError: 0.0269987042993307 | Loss: 0.0010078797502884 | Epoch: 96 | MeanAbsoluteEr

MeanAbsoluteError: 0.0434705018997192 | Loss: 0.0021727448329329 | Epoch: 115 | MeanAbsoluteE

MeanAbsoluteError: 0.0185597315430641 | Loss: 0.0006777943555407 | Epoch: 134 | MeanAbsoluteE

MeanAbsoluteError: 0.0134533401578665 | Loss: 0.0003910650771822 | Epoch: 153 | MeanAbsoluteE

MeanAbsoluteError: 0.0147619433701038 | Loss: 0.0004536946653388 | Epoch: 172 | MeanAbsoluteE

MeanAbsoluteError: 0.0388214290142059 | Loss: 0.0019600290412966 | Epoch: 191 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.0152238653972745 | Loss: 0.0004005020570108 | Epoch: 210 | MeanAbsoluteE

MeanAbsoluteError: 0.0144106419757009 | Loss: 0.0004485251686818 | Epoch: 229 | MeanAbsoluteE

MeanAbsoluteError: 0.0121733881533146 | Loss: 0.0002302894051689 | Epoch: 248 | MeanAbsoluteE

spotPython tuning: 3.3469929719917434e-06 [##########] 100.00% Done...

<spotPython.spot.spot.Spot at 0x28f1b7220>
```

19.9 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section 14.9, see also the description in the documentation: Tensorboard.

19.10 Step 10: Results

After the hyperparameter tuning run is finished, the results can be analyzed as described in Section 14.10.

```
spot_tuner.plot_progress(log_y=False,
    filename="./figures/" + experiment_name+"_progress.png")
```

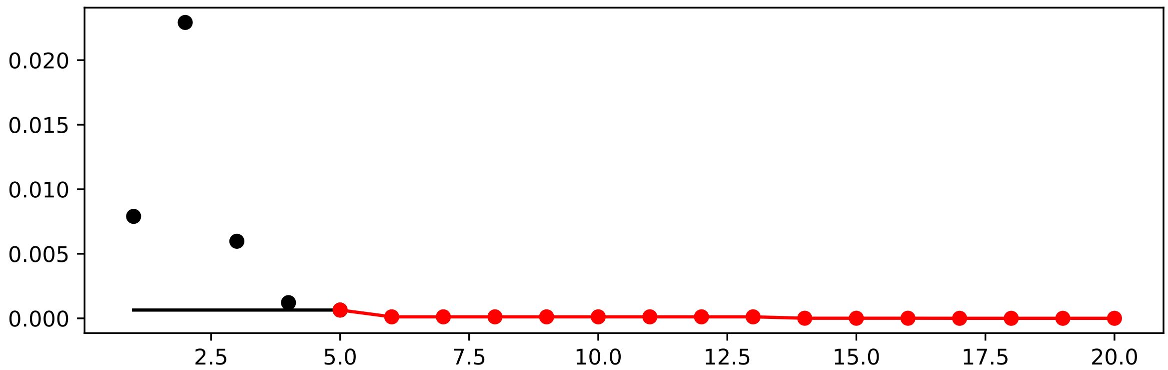


Figure 19.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

```
print(gen_design_table(fun_control=fun_control, spot=spot_tuner))
```

```
| name         | type   | default   |   lower |   upper |              tuned | transform
|--------------|--------|-----------|---------|---------|--------------------|---------------
| _L_in        | int    | 10        |    10.0 |    10.0 |               10.0 | None
| _L_out       | int    | 1         |     1.0 |     1.0 |                1.0 | None
| l1           | int    | 3         |     3.0 |     8.0 |                3.0 | transform_powe
| dropout_prob | float  | 0.01      |     0.0 |     0.9 |                0.0 | None
| lr_mult      | float  | 1.0       |     0.1 |    10.0 |  4.742898547325524 | None
| batch_size   | int    | 4         |     1.0 |     4.0 |                4.0 | transform_powe
| epochs       | int    | 4         |     2.0 |    16.0 |               13.0 | transform_powe
| k_folds      | int    | 1         |     1.0 |     1.0 |                1.0 | None
| patience     | int    | 2         |     3.0 |     7.0 |                7.0 | transform_powe
| optimizer    | factor | SGD       |     0.0 |     6.0 |                2.0 | None
| sgd_momentum | float  | 0.0       |     0.0 |     1.0 | 0.9262244566512092 | None
```

```
spot_tuner.plot_importance(threshold=0.025,
    filename="./figures/" + experiment_name+"_importance.png")
```

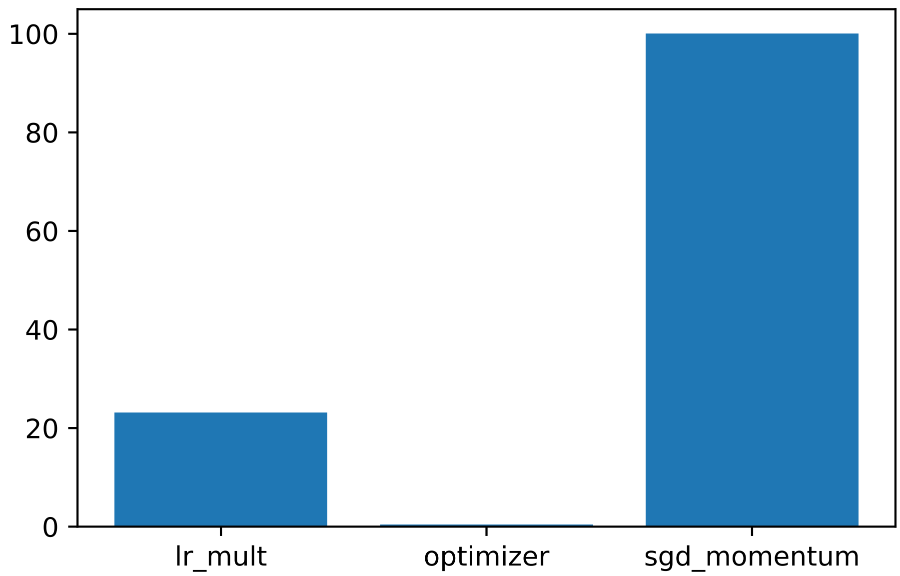


Figure 19.2: Variable importance plot, threshold 0.025.

19.10.1 Get the Tuned Architecture (SPOT Results)

```
from spotPython.hyperparameters.values import get_one_core_model_from_X
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
model_spot = get_one_core_model_from_X(X, fun_control)
model_spot
```

```
Net_lin_reg(
  (fc1): Linear(in_features=10, out_features=8, bias=True)
  (fc2): Linear(in_features=8, out_features=4, bias=True)
  (fc3): Linear(in_features=4, out_features=1, bias=True)
  (relu): ReLU()
  (softmax): Softmax(dim=1)
  (dropout1): Dropout(p=0.0, inplace=False)
  (dropout2): Dropout(p=0.0, inplace=False)
)
```

19.10.2 Evaluation of the Tuned Architecture

```
from spotPython.torch.traintest import (
    train_tuned,
    test_tuned,
    )
```

```
train_tuned(net=model_spot, train_dataset=train,
        loss_function=fun_control["loss_function"],
        metric=fun_control["metric_torch"],
        shuffle=True,
        device = fun_control["device"],
        path=None,
        task=fun_control["task"],)
```

```
Epoch: 1 | MeanAbsoluteError: 0.1210259869694710 | Loss: 0.0232771355658770 | Epoch: 2 | Mea

MeanAbsoluteError: 0.1038044840097427 | Loss: 0.0167114720189650 | Epoch: 8 | MeanAbsoluteEr

MeanAbsoluteError: 0.0086999181658030 | Loss: 0.0001652448006912 | Epoch: 24 | MeanAbsoluteE

MeanAbsoluteError: 0.0022539731580764 | Loss: 0.0000560129346354 | Epoch: 31 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.0020863686222583 | Loss: 0.0000346119437070 | Epoch: 47 | MeanAbsoluteE

MeanAbsoluteError: 0.0014666299102828 | Loss: 0.0000340013741771 | Epoch: 54 | MeanAbsoluteE

MeanAbsoluteError: 0.0015319933881983 | Loss: 0.0000251058726055 | Epoch: 70 | MeanAbsoluteE

MeanAbsoluteError: 0.0014555664965883 | Loss: 0.0000250080861226 | Epoch: 77 | MeanAbsoluteE

MeanAbsoluteError: 0.0032180577982217 | Loss: 0.0000272991795244 | Epoch: 93 | MeanAbsoluteE

MeanAbsoluteError: 0.0034448227379471 | Loss: 0.0000283073438224 | Epoch: 100 | MeanAbsolute

MeanAbsoluteError: 0.0023825149983168 | Loss: 0.0000229990641856 | Epoch: 116 | MeanAbsolute

MeanAbsoluteError: 0.0027838486712426 | Loss: 0.0000269139949954 | Epoch: 123 | MeanAbsolute

MeanAbsoluteError: 0.0041890218853951 | Loss: 0.0000334640817049 | Epoch: 139 | MeanAbsolute

MeanAbsoluteError: 0.0016274157678708 | Loss: 0.0000249198810408 | Epoch: 146 | MeanAbsolute

MeanAbsoluteError: 0.0031541502103209 | Loss: 0.0000295283362232 | Epoch: 162 | MeanAbsolute

MeanAbsoluteError: 0.0013988441787660 | Loss: 0.0000202680236701 | Epoch: 169 | MeanAbsolute

MeanAbsoluteError: 0.0017323890933767 | Loss: 0.0000223894536056 | Epoch: 185 | MeanAbsolute

MeanAbsoluteError: 0.0062805791385472 | Loss: 0.0000513286436456 | Epoch: 192 | MeanAbsolute

MeanAbsoluteError: 0.0017683045007288 | Loss: 0.0000198729687841 | Epoch: 208 | MeanAbsolute

MeanAbsoluteError: 0.0016275194939226 | Loss: 0.0000206559946575 | Epoch: 215 | MeanAbsolute

MeanAbsoluteError: 0.0037519137840718 | Loss: 0.0000315591232241 | Epoch: 231 | MeanAbsolute

MeanAbsoluteError: 0.0015745967393741 | Loss: 0.0000194911521200 | Epoch: 238 | MeanAbsolute

MeanAbsoluteError: 0.0014140426646918 | Loss: 0.0000184702457504 | Epoch: 254 | MeanAbsolute
```

```
MeanAbsoluteError: 0.0024122903123498 | Loss: 0.0000216886252187 | Epoch: 261 | MeanAbsoluteE

MeanAbsoluteError: 0.0029661676380783 | Loss: 0.0000223931743963 | Epoch: 277 | MeanAbsoluteE

MeanAbsoluteError: 0.0038125538267195 | Loss: 0.0000291012619480 | Epoch: 284 | MeanAbsoluteE

MeanAbsoluteError: 0.0011782477376983 | Loss: 0.0000160741511733 | Epoch: 300 | MeanAbsoluteE

MeanAbsoluteError: 0.0030522758606821 | Loss: 0.0000275352148683 | Epoch: 307 | MeanAbsoluteE

MeanAbsoluteError: 0.0019985507242382 | Loss: 0.0000184933844354 | Epoch: 323 | MeanAbsoluteE

MeanAbsoluteError: 0.0022178534418344 | Loss: 0.0000195868506604 | Epoch: 330 | MeanAbsoluteE

MeanAbsoluteError: 0.0012014196254313 | Loss: 0.0000161360356940 | Epoch: 346 | MeanAbsoluteE

MeanAbsoluteError: 0.0016414474230260 | Loss: 0.0000182519480452 | Epoch: 353 | MeanAbsoluteE

MeanAbsoluteError: 0.0012959231389686 | Loss: 0.0000144424530953 | Epoch: 369 | MeanAbsoluteE

MeanAbsoluteError: 0.0039915204979479 | Loss: 0.0000289531255571 | Epoch: 376 | MeanAbsoluteE

MeanAbsoluteError: 0.0042397510260344 | Loss: 0.0000302706241296 | Epoch: 392 | MeanAbsoluteE

MeanAbsoluteError: 0.0011300111655146 | Loss: 0.0000130624667708 | Epoch: 399 | MeanAbsoluteE

MeanAbsoluteError: 0.0017395407194272 | Loss: 0.0000143591699414 | Epoch: 415 | MeanAbsoluteE

MeanAbsoluteError: 0.0029490303713828 | Loss: 0.0000181601244549 | Epoch: 422 | MeanAbsoluteE

MeanAbsoluteError: 0.0014252760447562 | Loss: 0.0000122380411265 | Epoch: 438 | MeanAbsoluteE

MeanAbsoluteError: 0.0016586238052696 | Loss: 0.0000127458162189 | Epoch: 445 | MeanAbsoluteE

MeanAbsoluteError: 0.0052740485407412 | Loss: 0.0000371474629662 | Epoch: 461 | MeanAbsoluteE

MeanAbsoluteError: 0.0021467676851898 | Loss: 0.0000130071898998 | Epoch: 468 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.0032979054376483 | Loss: 0.0000203272580907 | Epoch: 484 | MeanAbsoluteE

MeanAbsoluteError: 0.0036079371348023 | Loss: 0.0000211418782304 | Epoch: 491 | MeanAbsoluteE

MeanAbsoluteError: 0.0019164363620803 | Loss: 0.0000106521777908 | Epoch: 507 | MeanAbsoluteE

MeanAbsoluteError: 0.0045759170316160 | Loss: 0.0000292285830686 | Epoch: 514 | MeanAbsoluteE

MeanAbsoluteError: 0.0070252656005323 | Loss: 0.0000535245064958 | Epoch: 530 | MeanAbsoluteE

MeanAbsoluteError: 0.0025557882618159 | Loss: 0.0000136797699796 | Epoch: 537 | MeanAbsoluteE

MeanAbsoluteError: 0.0017304082866758 | Loss: 0.0000104411718008 | Epoch: 553 | MeanAbsoluteE

MeanAbsoluteError: 0.0024793886113912 | Loss: 0.0000114748644304 | Epoch: 560 | MeanAbsoluteE

MeanAbsoluteError: 0.0012235768372193 | Loss: 0.0000072064945689 | Epoch: 576 | MeanAbsoluteE

MeanAbsoluteError: 0.0017975006485358 | Loss: 0.0000086525661501 | Epoch: 583 | MeanAbsoluteE

MeanAbsoluteError: 0.0013998147333041 | Loss: 0.0000075772047156 | Epoch: 599 | MeanAbsoluteE

MeanAbsoluteError: 0.0016223378479481 | Loss: 0.0000080170306570 | Epoch: 606 | MeanAbsoluteE

MeanAbsoluteError: 0.0010471467394382 | Loss: 0.0000059147427234 | Epoch: 622 | MeanAbsoluteE

MeanAbsoluteError: 0.0038996362127364 | Loss: 0.0000203398832654 | Epoch: 629 | MeanAbsoluteE

MeanAbsoluteError: 0.0014610006473958 | Loss: 0.0000065261032205 | Epoch: 645 | MeanAbsoluteE

MeanAbsoluteError: 0.0012839792761952 | Loss: 0.0000064887735609 | Epoch: 652 | MeanAbsoluteE

MeanAbsoluteError: 0.0030796492937952 | Loss: 0.0000145015428404 | Epoch: 668 | MeanAbsoluteE

MeanAbsoluteError: 0.0014493229100481 | Loss: 0.0000057345796449 | Epoch: 675 | MeanAbsoluteE

MeanAbsoluteError: 0.0014254114357755 | Loss: 0.0000062594065962 | Epoch: 691 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.0010765301994979 | Loss: 0.0000049606298376 | Epoch: 698 | MeanAbsolute
MeanAbsoluteError: 0.0019905383232981 | Loss: 0.0000074284228527 | Epoch: 714 | MeanAbsolute
MeanAbsoluteError: 0.0013641632394865 | Loss: 0.0000054332684481 | Epoch: 721 | MeanAbsolute
MeanAbsoluteError: 0.0087472656741738 | Loss: 0.0000828971150392 | Epoch: 737 | MeanAbsolute
MeanAbsoluteError: 0.0012725453125313 | Loss: 0.0000052797036843 | Epoch: 744 | MeanAbsolute
MeanAbsoluteError: 0.0027259869966656 | Loss: 0.0000108376517394 | Epoch: 760 | MeanAbsolute
MeanAbsoluteError: 0.0009826081804931 | Loss: 0.0000045526832800 | Epoch: 767 | MeanAbsolute
MeanAbsoluteError: 0.0016977831255645 | Loss: 0.0000058103874404 | Epoch: 783 | MeanAbsolute
MeanAbsoluteError: 0.0008775054593571 | Loss: 0.0000041182479510 | Epoch: 790 | MeanAbsolute
MeanAbsoluteError: 0.0014447929570451 | Loss: 0.0000054077370650 | Epoch: 806 | MeanAbsolute
MeanAbsoluteError: 0.0022551838774234 | Loss: 0.0000082007583459 | Epoch: 813 | MeanAbsolute
MeanAbsoluteError: 0.0020377174951136 | Loss: 0.0000067337236224 | Epoch: 829 | MeanAbsolute
MeanAbsoluteError: 0.0056775873526931 | Loss: 0.0000349798318846 | Epoch: 836 | MeanAbsolute
MeanAbsoluteError: 0.0008930975454859 | Loss: 0.0000028807081453 | Epoch: 852 | MeanAbsolute
MeanAbsoluteError: 0.0009378574322909 | Loss: 0.0000030204312853 | Epoch: 859 | MeanAbsolute
MeanAbsoluteError: 0.0021047412883490 | Loss: 0.0000069864005414 | Epoch: 875 | MeanAbsolute
MeanAbsoluteError: 0.0025349468924105 | Loss: 0.0000083534995802 | Epoch: 882 | MeanAbsolute
MeanAbsoluteError: 0.0009852692019194 | Loss: 0.0000028175826895 | Epoch: 898 | MeanAbsolute
MeanAbsoluteError: 0.0032205421011895 | Loss: 0.0000126960070415 | Epoch: 905 | MeanAbsolute
```

```
MeanAbsoluteError: 0.0019589774310589 | Loss: 0.0000058548103329 | Epoch: 921 | MeanAbsoluteE
MeanAbsoluteError: 0.0012278319336474 | Loss: 0.0000032181492615 | Epoch: 928 | MeanAbsoluteE
MeanAbsoluteError: 0.0016483683139086 | Loss: 0.0000047091259603 | Epoch: 944 | MeanAbsoluteE
MeanAbsoluteError: 0.0045855720527470 | Loss: 0.0000231344334009 | Epoch: 951 | MeanAbsoluteE
MeanAbsoluteError: 0.0009258669451810 | Loss: 0.0000023021689619 | Epoch: 967 | MeanAbsoluteE
MeanAbsoluteError: 0.0032089969608933 | Loss: 0.0000118345888040 | Epoch: 974 | MeanAbsoluteE
MeanAbsoluteError: 0.0035998192615807 | Loss: 0.0000156191096099 | Epoch: 990 | MeanAbsoluteE
MeanAbsoluteError: 0.0009243571548723 | Loss: 0.0000021139463797 | Epoch: 997 | MeanAbsoluteE
MeanAbsoluteError: 0.0022629154846072 | Loss: 0.0000067764048206 | Epoch: 1013 | MeanAbsolut
MeanAbsoluteError: 0.0008390324073844 | Loss: 0.0000017933644472 | Epoch: 1020 | MeanAbsolut
MeanAbsoluteError: 0.0023239566944540 | Loss: 0.0000071127543187 | Epoch: 1036 | MeanAbsolut
MeanAbsoluteError: 0.0035526501014829 | Loss: 0.0000145657700398 | Epoch: 1043 | MeanAbsolut
MeanAbsoluteError: 0.0007767161005177 | Loss: 0.0000015227647892 | Epoch: 1059 | MeanAbsolut
MeanAbsoluteError: 0.0058957566507161 | Loss: 0.0000357736469979 | Epoch: 1066 | MeanAbsolut
MeanAbsoluteError: 0.0016823011683300 | Loss: 0.0000041354599752 | Epoch: 1082 | MeanAbsolut
MeanAbsoluteError: 0.0009581772610545 | Loss: 0.0000018923689709 | Epoch: 1089 | MeanAbsolut
MeanAbsoluteError: 0.0033971278462559 | Loss: 0.0000131101764725 | Epoch: 1105 | MeanAbsolut
MeanAbsoluteError: 0.0018247729167342 | Loss: 0.0000045002373458 | Epoch: 1112 | MeanAbsolut
MeanAbsoluteError: 0.0030891210772097 | Loss: 0.0000113980833559 | Epoch: 1128 | MeanAbsolut
```

MeanAbsoluteError: 0.0036798783112317 | Loss: 0.0000146905810509 | Epoch: 1135 | MeanAbsolut

MeanAbsoluteError: 0.0015344627900049 | Loss: 0.0000034508210128 | Epoch: 1151 | MeanAbsolut

MeanAbsoluteError: 0.0008539916016161 | Loss: 0.0000014621217019 | Epoch: 1158 | MeanAbsolut

MeanAbsoluteError: 0.0016668232856318 | Loss: 0.0000039265098562 | Epoch: 1174 | MeanAbsolut

MeanAbsoluteError: 0.0010347056668252 | Loss: 0.0000018270739648 | Epoch: 1181 | MeanAbsolut

MeanAbsoluteError: 0.0029952884651721 | Loss: 0.0000106032284748 | Epoch: 1197 | MeanAbsolut

MeanAbsoluteError: 0.0012924659531564 | Loss: 0.0000027291306543 | Epoch: 1204 | MeanAbsolut

MeanAbsoluteError: 0.0023538593668491 | Loss: 0.0000074136541547 | Epoch: 1220 | MeanAbsolut

MeanAbsoluteError: 0.0016146614216268 | Loss: 0.0000036241362259 | Epoch: 1227 | MeanAbsolut

MeanAbsoluteError: 0.0011685898061842 | Loss: 0.0000022362305602 | Epoch: 1243 | MeanAbsolut

MeanAbsoluteError: 0.0027819366659969 | Loss: 0.0000087184545775 | Epoch: 1250 | MeanAbsolut

MeanAbsoluteError: 0.0044381795451045 | Loss: 0.0000218560168210 | Epoch: 1266 | MeanAbsolut

MeanAbsoluteError: 0.0019369859946892 | Loss: 0.0000050546581956 | Epoch: 1273 | MeanAbsolut

MeanAbsoluteError: 0.0014076898805797 | Loss: 0.0000028437563539 | Epoch: 1289 | MeanAbsolut

MeanAbsoluteError: 0.0009319037199020 | Loss: 0.0000014705289782 | Epoch: 1296 | MeanAbsolut

MeanAbsoluteError: 0.0032844021916389 | Loss: 0.0000123849197160 | Epoch: 1312 | MeanAbsolut

MeanAbsoluteError: 0.0017568019684404 | Loss: 0.0000040894341103 | Epoch: 1319 | MeanAbsolut

MeanAbsoluteError: 0.0042039970867336 | Loss: 0.0000191042233717 | Epoch: 1335 | MeanAbsolut

MeanAbsoluteError: 0.0019678811077029 | Loss: 0.0000052328829090 | Epoch: 1342 | MeanAbsolut

```
MeanAbsoluteError: 0.0015255718026310 | Loss: 0.0000034415114789 | Epoch: 1358 | MeanAbsolute

MeanAbsoluteError: 0.0043608583509922 | Loss: 0.0000201670243673 | Epoch: 1365 | MeanAbsolute

MeanAbsoluteError: 0.0010193425696343 | Loss: 0.0000016601040898 | Epoch: 1381 | MeanAbsolute

MeanAbsoluteError: 0.0041489191353321 | Loss: 0.0000185182159744 | Epoch: 1388 | MeanAbsolute

MeanAbsoluteError: 0.0027672438882291 | Loss: 0.0000087215007146 | Epoch: 1404 | MeanAbsolute

MeanAbsoluteError: 0.0020285064820200 | Loss: 0.0000050860587156 | Epoch: 1411 | MeanAbsolute

MeanAbsoluteError: 0.0015447561163455 | Loss: 0.0000033176343574 | Epoch: 1427 | MeanAbsolute

MeanAbsoluteError: 0.0020212903618813 | Loss: 0.0000053171171610 | Epoch: 1434 | MeanAbsolute

MeanAbsoluteError: 0.0008231356623583 | Loss: 0.0000011612264613 | Epoch: 1450 | MeanAbsolute

MeanAbsoluteError: 0.0037299969699234 | Loss: 0.0000152483454306 | Epoch: 1457 | MeanAbsolute

MeanAbsoluteError: 0.0033589589875191 | Loss: 0.0000122884707468 | Epoch: 1473 | MeanAbsolute

MeanAbsoluteError: 0.0013620426179841 | Loss: 0.0000026549154738 | Epoch: 1480 | MeanAbsolute

MeanAbsoluteError: 0.0008827612036839 | Loss: 0.0000013519076890 | Epoch: 1496 | MeanAbsolute

MeanAbsoluteError: 0.0026830476708710 | Loss: 0.0000085862896795 | Epoch: 1503 | MeanAbsolute

MeanAbsoluteError: 0.0034279366955161 | Loss: 0.0000129260902452 | Epoch: 1519 | MeanAbsolute

MeanAbsoluteError: 0.0009452450321987 | Loss: 0.0000014347141539 | Epoch: 1526 | MeanAbsolute
Returned to Spot: Validation loss: 8.018029587363248e-06
```

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: 0.0024604238569736 | Loss: 0.0000069522670572 | Final evaluation: Validati
Final evaluation: Validation metric: 0.002460423856973648
----------------------------------------------
```

```
(6.952267057158679e-06, nan, tensor(0.0025))
```

19.10.3 Cross-validated Evaluations

- This is the evaluation that will be used in the comparison (evaluatecv has to be updated before, to get metric vlaues!):

```
from spotPython.torch.traintest import evaluate_cv
# modify k-kolds:
setattr(model_spot, "k_folds",  10)
df_eval, df_preds, df_metrics = evaluate_cv(net=model_spot,
            dataset=fun_control["data"],
            loss_function=fun_control["loss_function"],
            metric=fun_control["metric_torch"],
            task=fun_control["task"],
            writer=fun_control["writer"],
            writerId="model_spot_cv",
            device = fun_control["device"])
```

```
Fold: 1
Epoch: 1 | MeanAbsoluteError: 0.1401705741882324 | Loss: 0.0288906829165561 | Epoch: 2 | Mean

MeanAbsoluteError: 0.0773800909519196 | Loss: 0.0097666824502604 | Epoch: 9 | MeanAbsoluteEr

MeanAbsoluteError: 0.0237553827464581 | Loss: 0.0007459620246664 | Epoch: 13 |

MeanAbsoluteError: 0.0175641737878323 | Loss: 0.0004824190026349 | Epoch: 14 | MeanAbsoluteE
```

MeanAbsoluteError: 0.0006911534001119 | Loss: 0.0000007603685625 | Epoch: 22 | MeanAbsoluteE

MeanAbsoluteError: 0.0010346834314987 | Loss: 0.0000015946940428 | Epoch: 26 |

MeanAbsoluteError: 0.0009236836340278 | Loss: 0.0000012056611597 | Epoch: 27 | MeanAbsoluteE

Epoch: 35 | MeanAbsoluteError: 0.0010179067030549 | Loss: 0.0000015546235578 | Epoch: 36 | Me

MeanAbsoluteError: 0.0009997882880270 | Loss: 0.0000013125084389 | Epoch: 39 |

MeanAbsoluteError: 0.0006845414754935 | Loss: 0.0000007233550312 | Epoch: 40 | MeanAbsoluteE

MeanAbsoluteError: 0.0023277786094695 | Loss: 0.0000059690660237 | Epoch: 49 | MeanAbsoluteE

MeanAbsoluteError: 0.0021826382726431 | Loss: 0.0000052542571081 |

Epoch: 53 | MeanAbsoluteError: 0.0008218445000239 | Loss: 0.0000008936666721 | Epoch: 54 | Me

MeanAbsoluteError: 0.0008194085676223 | Loss: 0.0000009852011138 | Epoch: 63 | MeanAbsoluteE

MeanAbsoluteError: 0.0007879968034104 | Loss: 0.0000011380851187 | Epoch: 67 | MeanAbsoluteE

MeanAbsoluteError: 0.0034944827202708 | Loss: 0.0000133304477328 | Epoch: 77 | MeanAbsoluteE

MeanAbsoluteError: 0.0021925384644419 | Loss: 0.0000056814741843 | Epoch: 81 | MeanAbsoluteE

MeanAbsoluteError: 0.0011348756961524 | Loss: 0.0000020344690524 | Epoch: 91 | MeanAbsoluteE

MeanAbsoluteError: 0.0009794107172638 | Loss: 0.0000014368039030 | Epoch: 95 | MeanAbsoluteE

MeanAbsoluteError: 0.0006996981683187 | Loss: 0.0000007805454792 | Epoch: 105 | MeanAbsolute

MeanAbsoluteError: 0.0015448528574780 | Loss: 0.0000028751961249 | Epoch: 108 | MeanAbsolute

MeanAbsoluteError: 0.0040446463972330 | Loss: 0.0000178401014637 | Epoch: 118 | MeanAbsolute

Epoch: 121 | MeanAbsoluteError: 0.0017152839573100 | Loss: 0.0000039971924772 | Epoch: 122 |

MeanAbsoluteError: 0.0018335819477215 | Loss: 0.0000039838751685 | Epoch: 132 | MeanAbsoluteE

MeanAbsoluteError: 0.0008728884276934 | Loss: 0.0000013137799004 | Epoch: 135 | MeanAbsoluteE

MeanAbsoluteError: 0.0034140234347433 | Loss: 0.0000132857226812 | Epoch: 146 | MeanAbsoluteE

MeanAbsoluteError: 0.0017687580548227 | Loss: 0.0000038433445297 | Epoch: 149 | MeanAbsoluteE

MeanAbsoluteError: 0.0018011000938714 | Loss: 0.0000036997589632 | Epoch: 160 | MeanAbsoluteE

MeanAbsoluteError: 0.0032381427008659 | Loss: 0.0000109853116166 | Epoch: 163 | MeanAbsoluteE
Fold: 2
Epoch: 1 | MeanAbsoluteError: 0.1275269091129303 | Loss: 0.0251138807673539 | Epoch: 2 |

MeanAbsoluteError: 0.1218319237232208 | Loss: 0.0211715595131474 | Epoch: 3 | MeanAbsoluteEr

MeanAbsoluteError: 0.0521842017769814 | Loss: 0.0040504606274356 | Epoch: 8 | MeanAbsoluteEr

MeanAbsoluteError: 0.0020287099760026 | Loss: 0.0000060799749073 | Epoch: 17 | MeanAbsoluteE

MeanAbsoluteError: 0.0007140208617784 | Loss: 0.0000007768023644 | Epoch: 22 | MeanAbsoluteE

MeanAbsoluteError: 0.0006557934102602 | Loss: 0.0000006669086637 | Epoch: 31 | MeanAbsoluteE

MeanAbsoluteError: 0.0007625867146999 | Loss: 0.0000010366534704 | Epoch: 36 | MeanAbsoluteE

MeanAbsoluteError: 0.0006542815244757 | Loss: 0.0000007570368317 | Epoch: 45 | MeanAbsoluteE

MeanAbsoluteError: 0.0006465223268606 | Loss: 0.0000006856930952 | Epoch: 50 | MeanAbsoluteE

MeanAbsoluteError: 0.0006503672921099 | Loss: 0.0000006529980031 | Epoch: 59 | MeanAbsoluteE

MeanAbsoluteError: 0.0010212395573035 | Loss: 0.0000013901644625 | Epoch: 64 | MeanAbsoluteE

MeanAbsoluteError: 0.0020053903572261 | Loss: 0.0000048233915615 | Epoch: 73 | MeanAbsoluteE

MeanAbsoluteError: 0.0007433213177137 | Loss: 0.0000007875052209 | Epoch: 78 | MeanAbsoluteE

```
MeanAbsoluteError: 0.0009387992322445 | Loss: 0.0000013290602072 | Epoch: 87 | MeanAbsoluteEr

MeanAbsoluteError: 0.0024404742289335 | Loss: 0.0000068521575746 | Epoch: 91 | MeanAbsoluteEr

MeanAbsoluteError: 0.0013644009595737 | Loss: 0.0000023604951431 | Epoch: 100 | MeanAbsoluteE

MeanAbsoluteError: 0.0020973056089133 | Loss: 0.0000050886945116 | Epoch: 104 | MeanAbsoluteE

Epoch: 113 | MeanAbsoluteError: 0.0026600523851812 | Loss: 0.0000078336926370 | Epoch: 114 |

MeanAbsoluteError: 0.0028487837407738 | Loss: 0.0000092020076668 | Epoch: 118 | MeanAbsoluteE

MeanAbsoluteError: 0.0016409807140008 | Loss: 0.0000033863658960 | Epoch: 127 | MeanAbsoluteE

MeanAbsoluteError: 0.0008353420416825 | Loss: 0.0000011150531886 | Epoch: 132 | MeanAbsoluteE

MeanAbsoluteError: 0.0010213337372988 | Loss: 0.0000016031120660 | Epoch: 141 | MeanAbsoluteE

MeanAbsoluteError: 0.0011321391211823 | Loss: 0.0000017707562035 | Epoch: 146 | MeanAbsoluteE
Fold: 3
Epoch: 1 | MeanAbsoluteError: 0.1212447956204414 | Loss: 0.0215919774823955 | Epoch: 2 | Mear

MeanAbsoluteError: 0.0666354000568390 | Loss: 0.0065281722761158 | Epoch: 5 | MeanAbsoluteErr

MeanAbsoluteError: 0.0019026905065402 | Loss: 0.0000080300579027 | Epoch: 9 | MeanAbsoluteErr

MeanAbsoluteError: 0.0008105455199257 | Loss: 0.0000010049964203 | Epoch: 19 | MeanAbsoluteEr

MeanAbsoluteError: 0.0008292419370264 | Loss: 0.0000009577929835 | Epoch: 23 | MeanAbsoluteEr

MeanAbsoluteError: 0.0007236693636514 | Loss: 0.0000007346643664 | Epoch: 33 | MeanAbsoluteEr

MeanAbsoluteError: 0.0010086777620018 | Loss: 0.0000014770049412 | Epoch: 37 | MeanAbsoluteEr

MeanAbsoluteError: 0.0008456934010610 | Loss: 0.0000012067267871 | Epoch: 47 | MeanAbsoluteEr

MeanAbsoluteError: 0.0015344852581620 | Loss: 0.0000031949065747 | Epoch: 51 | MeanAbsoluteEr
```

```
MeanAbsoluteError: 0.0007581984973513 | Loss: 0.0000007737808971 | Epoch: 61 | MeanAbsoluteE

MeanAbsoluteError: 0.0013248748145998 | Loss: 0.0000024402432440 | Epoch: 65 | MeanAbsoluteE

MeanAbsoluteError: 0.0010528377024457 | Loss: 0.0000015199398758 | Epoch: 75 | MeanAbsoluteE

MeanAbsoluteError: 0.0032286914065480 | Loss: 0.0000111964908293 | Epoch: 79 | MeanAbsoluteE

MeanAbsoluteError: 0.0007717724656686 | Loss: 0.0000009706845958 | Epoch: 88 | MeanAbsoluteE

MeanAbsoluteError: 0.0016662755515426 | Loss: 0.0000035471675801 | Epoch: 93 | MeanAbsoluteE

MeanAbsoluteError: 0.0010982968378812 | Loss: 0.0000017259161008 | Epoch: 102 | MeanAbsolute

MeanAbsoluteError: 0.0010409463429824 | Loss: 0.0000016362836439 | Epoch: 107 | MeanAbsolute

MeanAbsoluteError: 0.0022291177883744 | Loss: 0.0000060060949961 | Epoch: 116 | MeanAbsolute

MeanAbsoluteError: 0.0008693505078554 | Loss: 0.0000011688986693 | Epoch: 121 | MeanAbsolute

MeanAbsoluteError: 0.0023544845171273 | Loss: 0.0000061840458849 | Epoch: 130 | MeanAbsolute

MeanAbsoluteError: 0.0021203013602644 | Loss: 0.0000049528813081 | Epoch: 135 | MeanAbsolute

MeanAbsoluteError: 0.0020092318300158 | Loss: 0.0000047693701422 | Epoch: 144 | MeanAbsolute
Fold: 4
Epoch: 1 |

MeanAbsoluteError: 0.1256001144647598 | Loss: 0.0233442708849907 | Epoch: 2 | MeanAbsoluteEr

MeanAbsoluteError: 0.0016969385324046 | Loss: 0.0000297371760131 | Epoch: 11 | MeanAbsoluteE

Epoch: 15 | MeanAbsoluteError: 0.0012181025231257 | Loss: 0.0000063752491956 | Epoch: 16 | Me

MeanAbsoluteError: 0.0007874658913352 | Loss: 0.0000010815034232 | Epoch: 25 | MeanAbsoluteE

MeanAbsoluteError: 0.0009594744187780 | Loss: 0.0000013075486030 | Epoch: 28 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.0009582399507053 | Loss: 0.0000013797891825 | Epoch: 38 | MeanAbsoluteE

Epoch: 41 | MeanAbsoluteError: 0.0007810621755198 | Loss: 0.0000009232577359 | Epoch: 42 | M

MeanAbsoluteError: 0.0011759689077735 | Loss: 0.0000018902346548 | Epoch: 52 | MeanAbsoluteE

MeanAbsoluteError: 0.0014742988860235 | Loss: 0.0000026905639905 | Epoch: 55 | MeanAbsoluteE

MeanAbsoluteError: 0.0036690775305033 | Loss: 0.0000145205207705 | Epoch: 65 | MeanAbsoluteE

MeanAbsoluteError: 0.0007350760861300 | Loss: 0.0000008172071944 | Epoch: 68 | MeanAbsoluteE

MeanAbsoluteError: 0.0014356418978423 | Loss: 0.0000026658061480 | Epoch: 78 | MeanAbsoluteE

MeanAbsoluteError: 0.0007138505461626 | Loss: 0.0000008655618282 | Epoch: 81 | MeanAbsoluteE

MeanAbsoluteError: 0.0008972698706202 | Loss: 0.0000010690676788 | Epoch: 91 | MeanAbsoluteE

MeanAbsoluteError: 0.0112434672191739 | Loss: 0.0001279642144384 | Epoch: 94 | MeanAbsoluteE

Epoch: 104 | MeanAbsoluteError: 0.0029411748982966 | Loss: 0.0000096012840913 | Epoch: 105 |

Epoch: 107 | MeanAbsoluteError: 0.0052107665687799 | Loss: 0.0000280045655277 | Epoch: 108 |

MeanAbsoluteError: 0.0013346108607948 | Loss: 0.0000024563828203 | Epoch: 118 | MeanAbsolute

MeanAbsoluteError: 0.0010528345592320 | Loss: 0.0000015357087802 | Epoch: 121 | MeanAbsolute

MeanAbsoluteError: 0.0012605478987098 | Loss: 0.0000023000794727 | Epoch: 131 | MeanAbsolute

MeanAbsoluteError: 0.0035456805489957 | Loss: 0.0000131434463973 | Epoch: 134 | MeanAbsolute

MeanAbsoluteError: 0.0018572479020804 | Loss: 0.0000041879245925 | Epoch: 144 | MeanAbsolute

MeanAbsoluteError: 0.0040658432990313 | Loss: 0.0000183328795954 | Epoch: 147 | MeanAbsolute

MeanAbsoluteError: 0.0013781749876216 | Loss: 0.0000025440665143 | Epoch: 157 | MeanAbsolute
```

```
MeanAbsoluteError: 0.0027778544463217 | Loss: 0.0000087670515703 | Epoch: 160 | MeanAbsoluteE

MeanAbsoluteError: 0.0011397862108424 | Loss: 0.0000017364098182 | Epoch: 170 | MeanAbsoluteE

MeanAbsoluteError: 0.0014641470042989 | Loss: 0.0000029508584508 | Early stopping at epoch 1
Fold: 5
Epoch: 1 | MeanAbsoluteError: 0.1341199278831482 | Loss: 0.0278661603640233 | Epoch: 2 | Mean

MeanAbsoluteError: 0.0008041304536164 | Loss: 0.0000008920714509 | Epoch: 11 | MeanAbsoluteEr

MeanAbsoluteError: 0.0008677789592184 | Loss: 0.0000015088613574 | Epoch: 14 | MeanAbsoluteEr

MeanAbsoluteError: 0.0006727272411808 | Loss: 0.0000006944474730 | Epoch: 25 | MeanAbsoluteEr

MeanAbsoluteError: 0.0008184796315618 | Loss: 0.0000010376006035 | Epoch: 28 | MeanAbsoluteEr

MeanAbsoluteError: 0.0007387894438580 | Loss: 0.0000008782775061 | Epoch: 38 | MeanAbsoluteEr

MeanAbsoluteError: 0.0009721347596496 | Loss: 0.0000012376119603 | Epoch: 41 | MeanAbsoluteEr

Epoch: 51 | MeanAbsoluteError: 0.0007879266049713 | Loss: 0.0000009775971778 | Epoch: 52 | Me

MeanAbsoluteError: 0.0008557504625060 | Loss: 0.0000011265201303 | Epoch: 55 | MeanAbsoluteEr

MeanAbsoluteError: 0.0010429425165057 | Loss: 0.0000017540738457 | Epoch: 65 | MeanAbsoluteEr

MeanAbsoluteError: 0.0009017419652082 | Loss: 0.0000012353302249 | Epoch: 69 | MeanAbsoluteEr

MeanAbsoluteError: 0.0007304614991881 | Loss: 0.0000008121438465 | Epoch: 78 | MeanAbsoluteEr

MeanAbsoluteError: 0.0019982582889497 | Loss: 0.0000051867161476 | Epoch: 82 | MeanAbsoluteEr

MeanAbsoluteError: 0.0043685832060874 | Loss: 0.0000204880107049 | Epoch: 92 | MeanAbsoluteEr

MeanAbsoluteError: 0.0031887784134597 | Loss: 0.0000119370702773 | Epoch: 96 | MeanAbsoluteEr

MeanAbsoluteError: 0.0012187439715490 | Loss: 0.0000019346234694 | Epoch: 105 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.0008503264398314 | Loss: 0.0000010767450850 | Epoch: 109 | MeanAbsoluteE

MeanAbsoluteError: 0.0022100848145783 | Loss: 0.0000059890170243 | Epoch: 119 | MeanAbsoluteE

MeanAbsoluteError: 0.0026359523180872 | Loss: 0.0000078452246401 | Epoch: 123 | MeanAbsoluteE

Epoch: 132 | MeanAbsoluteError: 0.0017066588625312 | Loss: 0.0000036819259159 | Epoch: 133 |

MeanAbsoluteError: 0.0024363398551941 | Loss: 0.0000069549872868 | Epoch: 136 | MeanAbsoluteE

MeanAbsoluteError: 0.0012961870525032 | Loss: 0.0000023045362890 | Epoch: 145 | MeanAbsoluteE

MeanAbsoluteError: 0.0026249112561345 | Loss: 0.0000080710971296 | Epoch: 149 | MeanAbsoluteE
Fold: 6
Epoch: 1 | MeanAbsoluteError: 0.1336473524570465 | Loss: 0.0276835563459567 | Epoch: 2 | Mea

MeanAbsoluteError: 0.0907206907868385 | Loss: 0.0125754815526307 | Epoch: 6 | MeanAbsoluteEr

MeanAbsoluteError: 0.0345026850700378 | Loss: 0.0018931587941812 | Epoch: 10 | MeanAbsoluteE

MeanAbsoluteError: 0.0011898803059012 | Loss: 0.0000028518087447 | Epoch: 19 | MeanAbsoluteE

MeanAbsoluteError: 0.0008515117224306 | Loss: 0.0000015512777704 | Epoch: 23 | MeanAbsoluteE

MeanAbsoluteError: 0.0007972165476531 | Loss: 0.0000011622501574 | Epoch: 32 | MeanAbsoluteE

MeanAbsoluteError: 0.0007602002588101 | Loss: 0.0000009226145526 | Epoch: 36 | MeanAbsoluteE

MeanAbsoluteError: 0.0009697804343887 | Loss: 0.0000013068494695 | Epoch: 45 | MeanAbsoluteE

MeanAbsoluteError: 0.0013763755559921 | Loss: 0.0000024716371822 | Epoch: 49 | MeanAbsoluteE

MeanAbsoluteError: 0.0009062201716006 | Loss: 0.0000011605257311 | Epoch: 58 | MeanAbsoluteE

MeanAbsoluteError: 0.0015150706749409 | Loss: 0.0000028546197492 | Epoch: 62 | MeanAbsoluteE

MeanAbsoluteError: 0.0008850943413563 | Loss: 0.0000014268295964 | Epoch: 71 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.0008310212870128 | Loss: 0.0000010463623405 | Epoch: 75 | MeanAbsoluteEr

MeanAbsoluteError: 0.0020150623749942 | Loss: 0.0000049856989790 | Epoch: 84 | MeanAbsoluteEr

MeanAbsoluteError: 0.0008958307444118 | Loss: 0.0000012160426926 | Epoch: 88 | MeanAbsoluteEr

MeanAbsoluteError: 0.0010872352868319 | Loss: 0.0000019627064246 | Epoch: 97 | MeanAbsoluteEr

MeanAbsoluteError: 0.0022573745809495 | Loss: 0.0000060716537778 | Epoch: 101 | MeanAbsoluteE

MeanAbsoluteError: 0.0011669625528157 | Loss: 0.0000019381636085 | Epoch: 110 | MeanAbsoluteE

MeanAbsoluteError: 0.0053503625094891 | Loss: 0.0000288261214467 | Epoch: 114 | MeanAbsoluteE

MeanAbsoluteError: 0.0009171689162031 | Loss: 0.0000011634891004 | Epoch: 123 | MeanAbsoluteE

MeanAbsoluteError: 0.0011599123245105 | Loss: 0.0000022105474337 | Epoch: 127 | MeanAbsoluteE

MeanAbsoluteError: 0.0008592121885158 | Loss: 0.0000010263741745 | Epoch: 136 | MeanAbsoluteE

MeanAbsoluteError: 0.0008251405670308 | Loss: 0.0000010259566596 | Epoch: 140 | MeanAbsoluteE

MeanAbsoluteError: 0.0022146103437990 | Loss: 0.0000058551711066 | Epoch: 149 | MeanAbsoluteE

MeanAbsoluteError: 0.0074297189712524 | Loss: 0.0000559491854801 | Epoch: 153 | MeanAbsoluteE

MeanAbsoluteError: 0.0027010089252144 | Loss: 0.0000085843325256 | Epoch: 162 | MeanAbsoluteE

MeanAbsoluteError: 0.0008831784944050 | Loss: 0.0000014416796392 | Epoch: 167 | MeanAbsoluteE

MeanAbsoluteError: 0.0013482858194038 | Loss: 0.0000022933868097 | Epoch: 175 | MeanAbsoluteE

MeanAbsoluteError: 0.0008698677411303 | Loss: 0.0000010991026004 | Epoch: 180 | MeanAbsoluteE

Epoch: 188 | MeanAbsoluteError: 0.0009416171233170 | Loss: 0.0000012753843391 | Epoch: 189 |

MeanAbsoluteError: 0.0008449642919004 | Loss: 0.0000010330378067 | Epoch: 194 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.0030690599232912 | Loss: 0.0000103894321260 | Early stopping at epoch 2C
Fold: 7
Epoch: 1 | MeanAbsoluteError: 0.1052020937204361 | Loss: 0.0158157053935741 | Epoch: 2 | Mear

MeanAbsoluteError: 0.0332100540399551 | Loss: 0.0015676429694785 | Epoch: 6 | MeanAbsoluteEr

MeanAbsoluteError: 0.0011294090654701 | Loss: 0.0000017438050658 | Epoch: 14 | MeanAbsoluteE

MeanAbsoluteError: 0.0008728299872018 | Loss: 0.0000011892214583 | Epoch: 20 | MeanAbsoluteE

MeanAbsoluteError: 0.0006999263423495 | Loss: 0.0000008820306821 | Epoch: 28 | MeanAbsoluteE

Epoch: 33 | MeanAbsoluteError: 0.0010195693466812 | Loss: 0.0000014622351614 | Epoch: 34 | Me

MeanAbsoluteError: 0.0008048696909100 | Loss: 0.0000011372521474 | Epoch: 41 | MeanAbsoluteE

Epoch: 46 | MeanAbsoluteError: 0.0012663902016357 | Loss: 0.0000022156496990 | Epoch: 47 | Me

Epoch: 54 | MeanAbsoluteError: 0.0027364660054445 | Loss: 0.0000089813001978 | Epoch: 55 | Me

MeanAbsoluteError: 0.0009089697850868 | Loss: 0.0000010995462714 | Epoch: 60 | MeanAbsoluteE

MeanAbsoluteError: 0.0008067876333371 | Loss: 0.0000011846218797 | Epoch: 68 | MeanAbsoluteE

Epoch: 73 | MeanAbsoluteError: 0.0007645866135135 | Loss: 0.0000009064134068 | Epoch: 74 | Me

Epoch: 81 | MeanAbsoluteError: 0.0038365691434592 | Loss: 0.0000159213889544 | Epoch: 82 | Me

MeanAbsoluteError: 0.0007330277585424 | Loss: 0.0000008517479598 | Epoch: 87 | MeanAbsoluteE

MeanAbsoluteError: 0.0010425288928673 | Loss: 0.0000017423564275 | Epoch: 95 | MeanAbsoluteE

MeanAbsoluteError: 0.0025835102424026 | Loss: 0.0000074046290268 | Epoch: 101 | MeanAbsolute

MeanAbsoluteError: 0.0021466943435371 | Loss: 0.0000058084324987 | Epoch: 109 | MeanAbsolute

MeanAbsoluteError: 0.0010840442264453 | Loss: 0.0000017967251357 | Epoch: 115 | MeanAbsolute
```

```
MeanAbsoluteError: 0.0033869827166200 | Loss: 0.0000120308156641 | Epoch: 123 | MeanAbsoluteE

MeanAbsoluteError: 0.0017346012173221 | Loss: 0.0000043487992311 | Epoch: 129 | MeanAbsoluteE

MeanAbsoluteError: 0.0018099199514836 | Loss: 0.0000047231462759 | Epoch: 137 | MeanAbsoluteE

Epoch: 142 | MeanAbsoluteError: 0.0009106831275858 | Loss: 0.0000012379367961 | Epoch: 143 |

Epoch: 150 | MeanAbsoluteError: 0.0009455553954467 | Loss: 0.0000013219960806 | Epoch: 151 |

MeanAbsoluteError: 0.0012263946700841 | Loss: 0.0000019040187778 | Epoch: 155 | MeanAbsoluteE
Fold: 8
Epoch: 1 | MeanAbsoluteError: 0.1433426141738892 | Loss: 0.0292781433090568 | Epoch: 2 | Mean

MeanAbsoluteError: 0.1342652142047882 | Loss: 0.0262951856212957 | Epoch: 4 | MeanAbsoluteEr

MeanAbsoluteError: 0.1324447691440582 | Loss: 0.0286753480987889 | Epoch: 9 | MeanAbsoluteEr

MeanAbsoluteError: 0.1322540640830994 | Loss: 0.0267146852399622 | Epoch: 17 | MeanAbsoluteE

MeanAbsoluteError: 0.1356780081987381 | Loss: 0.0271844249218702 | Epoch: 23 | MeanAbsoluteE

MeanAbsoluteError: 0.1331114619970322 | Loss: 0.0266668623579400 | Epoch: 31 | MeanAbsoluteE

MeanAbsoluteError: 0.1392946690320969 | Loss: 0.0291133169084787 | Epoch: 37 | MeanAbsoluteE

MeanAbsoluteError: 0.1341278702020645 | Loss: 0.0261373255135758 | Epoch: 45 | MeanAbsoluteE

MeanAbsoluteError: 0.1360089778900146 | Loss: 0.0276702551969460 | Epoch: 51 | MeanAbsoluteE

MeanAbsoluteError: 0.1327786594629288 | Loss: 0.0249720255711249 | Epoch: 59 | MeanAbsoluteE

MeanAbsoluteError: 0.1349911987781525 | Loss: 0.0290933373783316 | Epoch: 65 | MeanAbsoluteE

MeanAbsoluteError: 0.1353220641613007 | Loss: 0.0302636548876762 | Epoch: 73 | MeanAbsoluteE

MeanAbsoluteError: 0.1336453855037689 | Loss: 0.0250547066597002 | Epoch: 79 | MeanAbsoluteE
```

MeanAbsoluteError: 0.1355779618024826 | Loss: 0.0297566822596959 | Epoch: 87 | MeanAbsoluteE

MeanAbsoluteError: 0.1383795589208603 | Loss: 0.0282049777784518 | Epoch: 93 | MeanAbsoluteE

MeanAbsoluteError: 0.1324404627084732 | Loss: 0.0277330359177930 | Epoch: 101 | MeanAbsolute

Epoch: 106 | MeanAbsoluteError: 0.1382904648780823 | Loss: 0.0338723358831235 | Epoch: 107 |

MeanAbsoluteError: 0.1326576620340347 | Loss: 0.0266984658581870 | Epoch: 114 | MeanAbsolute

MeanAbsoluteError: 0.1352486610412598 | Loss: 0.0293138543409961 | Epoch: 119 | MeanAbsolute

MeanAbsoluteError: 0.1322540789842606 | Loss: 0.0316944779562099 | Epoch: 127 | MeanAbsolute

MeanAbsoluteError: 0.1332598030567169 | Loss: 0.0264483205974102 | Epoch: 132 | MeanAbsolute

Epoch: 140 | MeanAbsoluteError: 0.1346901655197144 | Loss: 0.0280776634546263 | Epoch: 141 |

MeanAbsoluteError: 0.1328521668910980 | Loss: 0.0283182625259672 | Epoch: 146 | MeanAbsolute

MeanAbsoluteError: 0.1368778198957443 | Loss: 0.0261353936844638 | Epoch: 154 | MeanAbsolute

MeanAbsoluteError: 0.1350857764482498 | Loss: 0.0257605553737708 | Epoch: 160 | MeanAbsolute

Epoch: 167 | MeanAbsoluteError: 0.1324919760227203 | Loss: 0.0283804062221731 | Epoch: 168 |

Epoch: 173 | MeanAbsoluteError: 0.1322869658470154 | Loss: 0.0262594188430480 | Epoch: 174 |

MeanAbsoluteError: 0.1324746608734131 | Loss: 0.0256296743505767 | Epoch: 180 | MeanAbsolute

MeanAbsoluteError: 0.1324398070573807 | Loss: 0.0263632365635463 | Epoch: 187 | MeanAbsolute

MeanAbsoluteError: 0.1324222385883331 | Loss: 0.0262428697730814 | Epoch: 194 | MeanAbsolute

MeanAbsoluteError: 0.1346070915460587 | Loss: 0.0262067102427994 | Epoch: 201 | MeanAbsolute

MeanAbsoluteError: 0.1389878392219543 | Loss: 0.0317239795944520 | Epoch: 208 | MeanAbsolute

```
Epoch: 214 | MeanAbsoluteError: 0.1335689872503281 | Loss: 0.0281970735107149 | Epoch: 215 |

MeanAbsoluteError: 0.1326252818107605 | Loss: 0.0291642235325915 | Epoch: 222 | MeanAbsoluteE

MeanAbsoluteError: 0.1324778348207474 | Loss: 0.0264026823320559 | Epoch: 228 | MeanAbsoluteE

Epoch: 235 | MeanAbsoluteError: 0.1343051642179489 | Loss: 0.0307509579828807 | Epoch: 236 |

Epoch: 241 | MeanAbsoluteError: 0.1322725713253021 | Loss: 0.0296348727175168 | Epoch: 242 |

MeanAbsoluteError: 0.1324019581079483 | Loss: 0.0248638660913067 | Epoch: 249 | MeanAbsoluteE
Fold: 9
Epoch: 1 | MeanAbsoluteError: 0.1028268039226532 | Loss: 0.0177168450983507 | Epoch: 2 | Mean

MeanAbsoluteError: 0.0929241925477982 | Loss: 0.0164195712921875 | Epoch: 4 | MeanAbsoluteEr

Epoch: 12 | MeanAbsoluteError: 0.0504019819200039 | Loss: 0.0038485082664660 | Epoch: 13 | Me

MeanAbsoluteError: 0.0219327043741941 | Loss: 0.0007094063455172 | Epoch: 18 | MeanAbsoluteE

MeanAbsoluteError: 0.0041630747728050 | Loss: 0.0000248750406432 | Epoch: 26 | MeanAbsoluteE

MeanAbsoluteError: 0.0012191210407764 | Loss: 0.0000021560218296 | Epoch: 32 | MeanAbsoluteE

MeanAbsoluteError: 0.0006714271148667 | Loss: 0.0000007052837369 | Epoch: 39 | MeanAbsoluteE

MeanAbsoluteError: 0.0011639861622825 | Loss: 0.0000018008584643 | Epoch: 46 | MeanAbsoluteE

MeanAbsoluteError: 0.0008167575579137 | Loss: 0.0000010143358021 | Epoch: 53 | MeanAbsoluteE

MeanAbsoluteError: 0.0008935870137066 | Loss: 0.0000011561439806 | Epoch: 60 | MeanAbsoluteE

MeanAbsoluteError: 0.0013254167279229 | Loss: 0.0000021793287845 | Epoch: 67 | MeanAbsoluteE

MeanAbsoluteError: 0.0006606917013414 | Loss: 0.0000006968045463 | Epoch: 73 | MeanAbsoluteE

MeanAbsoluteError: 0.0006987686501816 | Loss: 0.0000007266255843 | Epoch: 78 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.0012517778668553 | Loss: 0.0000020578226148 | Epoch: 87 | MeanAbsoluteE

MeanAbsoluteError: 0.0007402481278405 | Loss: 0.0000007943099080 | Epoch: 92 | MeanAbsoluteE

MeanAbsoluteError: 0.0025111085269600 | Loss: 0.0000071442849341 | Epoch: 101 | MeanAbsolute

MeanAbsoluteError: 0.0010711775394157 | Loss: 0.0000017391663830 | Epoch: 106 | MeanAbsolute

MeanAbsoluteError: 0.0010902507929131 | Loss: 0.0000015704082768 | Epoch: 115 | MeanAbsolute

MeanAbsoluteError: 0.0016297106631100 | Loss: 0.0000033652057872 | Epoch: 120 | MeanAbsolute

MeanAbsoluteError: 0.0015441904542968 | Loss: 0.0000031639095011 | Epoch: 129 | MeanAbsolute

MeanAbsoluteError: 0.0007531912415288 | Loss: 0.0000008669014976 | Epoch: 134 | MeanAbsolute

MeanAbsoluteError: 0.0020602797158062 | Loss: 0.0000049586764622 | Epoch: 143 | MeanAbsolute

MeanAbsoluteError: 0.0010382877662778 | Loss: 0.0000016202389556 | Epoch: 148 | MeanAbsolute

MeanAbsoluteError: 0.0013918275944889 | Loss: 0.0000025767596818 | Epoch: 156 | MeanAbsolute

MeanAbsoluteError: 0.0034683891572058 | Loss: 0.0000129177408650 | Epoch: 162 | MeanAbsolute

MeanAbsoluteError: 0.0020796542521566 | Loss: 0.0000052714967751 | Epoch: 170 | MeanAbsolute

MeanAbsoluteError: 0.0031262256670743 | Loss: 0.0000114306610288 | Early stopping at epoch 1
Fold: 10
Epoch: 1 | MeanAbsoluteError: 0.1344988793134689 | Loss: 0.0314739447619234 | Epoch: 2 | Mea

MeanAbsoluteError: 0.0414936207234859 | Loss: 0.0029552647444819 | Epoch: 8 | MeanAbsoluteEr

MeanAbsoluteError: 0.0022353110834956 | Loss: 0.0000822076200068 | Epoch: 14 | MeanAbsoluteE

MeanAbsoluteError: 0.0016148426802829 | Loss: 0.0001945993686263 | Epoch: 21 | MeanAbsoluteE

MeanAbsoluteError: 0.0023917995858938 | Loss: 0.0000397926414410 | Epoch: 27 | MeanAbsoluteE
```

MeanAbsoluteError: 0.0012229640269652 | Loss: 0.0000262392768125 | Epoch: 34 | MeanAbsoluteE

MeanAbsoluteError: 0.0012120216852054 | Loss: 0.0000183334427431 | Epoch: 41 | MeanAbsoluteE

MeanAbsoluteError: 0.0010731224901974 | Loss: 0.0000119559133184 | Epoch: 48 | MeanAbsoluteE

MeanAbsoluteError: 0.0021527092903852 | Loss: 0.0000124688056076 | Epoch: 54 | MeanAbsoluteE

MeanAbsoluteError: 0.0033273366279900 | Loss: 0.0000162877364086 | Epoch: 61 | MeanAbsoluteE

MeanAbsoluteError: 0.0020738877356052 | Loss: 0.0000073986676788 | Epoch: 67 | MeanAbsoluteE

Epoch: 74 | MeanAbsoluteError: 0.0016165536362678 | Loss: 0.0000111695036529 | Epoch: 75 | M

MeanAbsoluteError: 0.0011259140446782 | Loss: 0.0000031473370297 | Epoch: 81 | MeanAbsoluteE

MeanAbsoluteError: 0.0018445742316544 | Loss: 0.0000049295898147 | Epoch: 88 | MeanAbsoluteE

MeanAbsoluteError: 0.0021692554000765 | Loss: 0.0000065271037784 | Epoch: 95 | MeanAbsoluteE

MeanAbsoluteError: 0.0017415547044948 | Loss: 0.0000043936292968 | Epoch: 102 | MeanAbsolute

MeanAbsoluteError: 0.0007939951610751 | Loss: 0.0000016274863859 | Epoch: 109 | MeanAbsolute

MeanAbsoluteError: 0.0020091242622584 | Loss: 0.0000056183340088 | Epoch: 116 | MeanAbsolute

MeanAbsoluteError: 0.0034446739591658 | Loss: 0.0000136614840553 | Epoch: 123 | MeanAbsolute

MeanAbsoluteError: 0.0011290680849925 | Loss: 0.0000024867764036 | Epoch: 130 | MeanAbsolute

Epoch: 136 | MeanAbsoluteError: 0.0023947025183588 | Loss: 0.0000067483998204 | Epoch: 137 |

MeanAbsoluteError: 0.0037261610850692 | Loss: 0.0000149070054769 | Epoch: 143 | MeanAbsolute

MeanAbsoluteError: 0.0014031326863915 | Loss: 0.0000026585932054 | Epoch: 150 | MeanAbsolute

MeanAbsoluteError: 0.0014972911449149 | Loss: 0.0000031981393736 | Epoch: 157 | MeanAbsolute

```
MeanAbsoluteError: 0.0007987638819031 | Loss: 0.0000012998346181 | Epoch: 164 | MeanAbsoluteE

MeanAbsoluteError: 0.0012702223611996 | Loss: 0.0000026891926268 | Epoch: 171 | MeanAbsoluteE

MeanAbsoluteError: 0.0020826528780162 | Loss: 0.0000056878415567 | Epoch: 178 | MeanAbsoluteE

MeanAbsoluteError: 0.0011505387956277 | Loss: 0.0000018678322574 | Epoch: 185 | MeanAbsoluteE

MeanAbsoluteError: 0.0038439023774117 | Loss: 0.0000169013233712 | Epoch: 192 | MeanAbsoluteE

MeanAbsoluteError: 0.0019588014110923 | Loss: 0.0000044102914671 | Epoch: 199 | MeanAbsoluteE

MeanAbsoluteError: 0.0007157339132391 | Loss: 0.0000009412878431 | Epoch: 206 | MeanAbsoluteE

MeanAbsoluteError: 0.0065248296596110 | Loss: 0.0000436128897750 | Epoch: 213 | MeanAbsoluteE

MeanAbsoluteError: 0.0024420216213912 | Loss: 0.0000068249345791 | Epoch: 220 | MeanAbsoluteE

Epoch: 226 | MeanAbsoluteError: 0.0017238968284801 | Loss: 0.0000036150736053 | Epoch: 227 |

MeanAbsoluteError: 0.0016392316902056 | Loss: 0.0000031437428691 | Epoch: 233 | MeanAbsoluteE

MeanAbsoluteError: 0.0016275927191600 | Loss: 0.0000037823335788 | Epoch: 239 | MeanAbsoluteE

MeanAbsoluteError: 0.0031377850100398 | Loss: 0.0000112799037067 | Epoch: 247 | MeanAbsoluteE

MeanAbsoluteError: 0.0022801468148828 | Loss: 0.0000059790752727 | Epoch: 253 | MeanAbsoluteE

MeanAbsoluteError: 0.0016342672752216 | Loss: 0.0000032637394725 | Epoch: 261 | MeanAbsoluteE

MeanAbsoluteError: 0.0050797159783542 | Loss: 0.0000285996403753 | Epoch: 267 | MeanAbsoluteE

MeanAbsoluteError: 0.0009863822488114 | Loss: 0.0000013571829913 | Epoch: 275 | MeanAbsoluteE

MeanAbsoluteError: 0.0007434940198436 | Loss: 0.0000008393508243 | Epoch: 281 | MeanAbsoluteE

MeanAbsoluteError: 0.0010949376737699 | Loss: 0.0000017664207722 | Epoch: 289 | MeanAbsoluteE
```

```
MeanAbsoluteError: 0.0033483987208456 | Loss: 0.0000128341449194 | Epoch: 295 | MeanAbsoluteE

MeanAbsoluteError: 0.0011632055975497 | Loss: 0.0000018598525133 | Epoch: 303 | MeanAbsoluteE

MeanAbsoluteError: 0.0019101544748992 | Loss: 0.0000046799927337 | Epoch: 308 | MeanAbsoluteE

MeanAbsoluteError: 0.0009822690626606 | Loss: 0.0000015711811849 | Epoch: 317 | MeanAbsoluteE

MeanAbsoluteError: 0.0026310062967241 | Loss: 0.0000075920599686 | Epoch: 322 | MeanAbsoluteE

MeanAbsoluteError: 0.0035134262870997 | Loss: 0.0000128526158473 | Epoch: 331 | MeanAbsoluteE

MeanAbsoluteError: 0.0007935780449770 | Loss: 0.0000009645800349 | Epoch: 336 | MeanAbsoluteE

MeanAbsoluteError: 0.0015784434508532 | Loss: 0.0000033874282508 | Epoch: 345 | MeanAbsoluteE

MeanAbsoluteError: 0.0020061752293259 | Loss: 0.0000048761610612 | Epoch: 350 | MeanAbsoluteE

MeanAbsoluteError: 0.0012507305946201 | Loss: 0.0000021697052424 | Epoch: 359 | MeanAbsoluteE

MeanAbsoluteError: 0.0047209970653057 | Loss: 0.0000241212928813 | Epoch: 364 | MeanAbsoluteE

MeanAbsoluteError: 0.0025903957430273 | Loss: 0.0000079177606754 | Epoch: 373 | MeanAbsoluteE

MeanAbsoluteError: 0.0010689548216760 | Loss: 0.0000014969466438 | Epoch: 378 | MeanAbsoluteE

MeanAbsoluteError: 0.0015782538102940 | Loss: 0.0000030711259699 | Epoch: 387 | MeanAbsoluteE

MeanAbsoluteError: 0.0025875689461827 | Loss: 0.0000076213744121 | Epoch: 392 | MeanAbsoluteE

MeanAbsoluteError: 0.0007031944114715 | Loss: 0.0000007623856553 | Early stopping at epoch 39
```

```
metric_name = type(fun_control["metric_torch"]).__name__
print(f"loss: {df_eval}, Cross-validated {metric_name}: {df_metrics}")
```

```
loss: 0.0026007111247646924, Cross-validated MeanAbsoluteError: 0.015066762454807758
```

19.10.4 Detailed Hyperparameter Plots

```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

```
lr_mult:  23.110738739570817
optimizer:  0.4116663641576298
sgd_momentum:  100.0
```

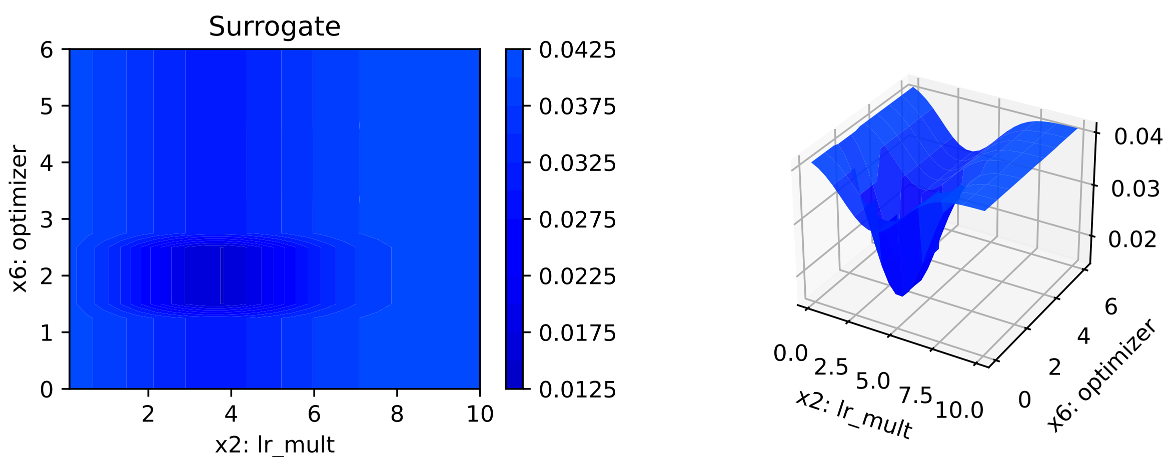
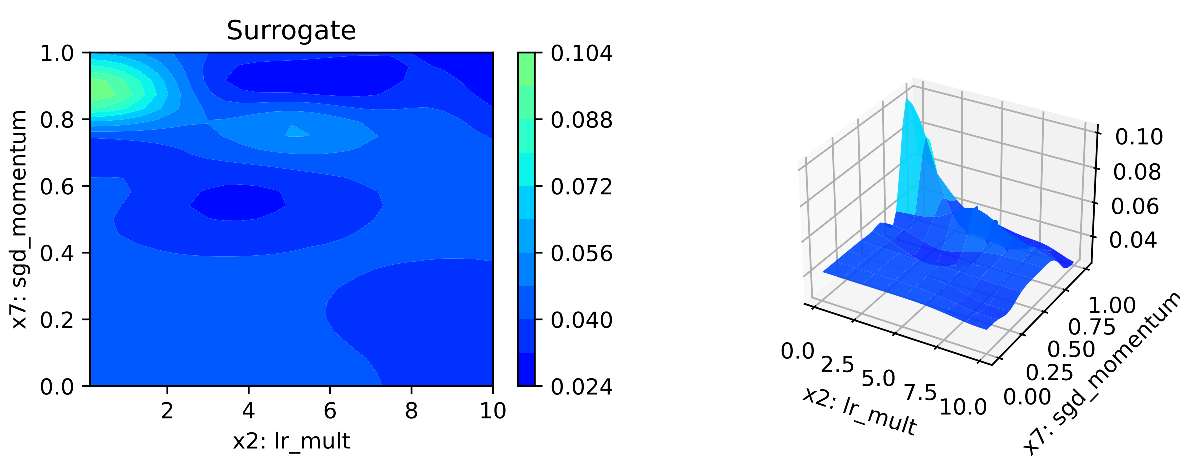
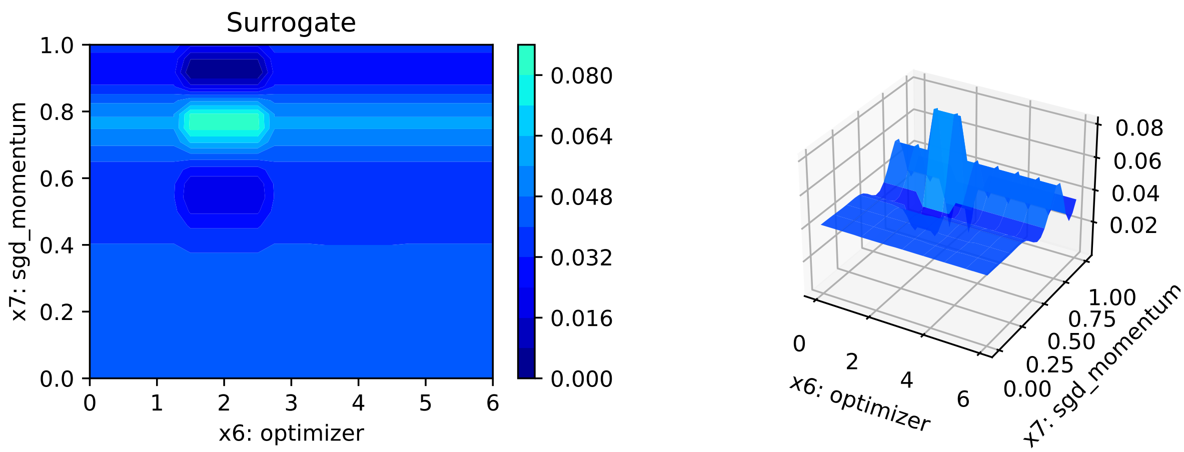


Figure 19.3: Contour plots.





19.10.5 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

```
Unable to display output for mime type(s): text/html
```

Parallel coordinates plots

```
Unable to display output for mime type(s): text/html
```

19.11 Summary and Outlook

This tutorial presents the hyperparameter tuning open source software `spotPython` for `PyTorch`. Some of the advantages of `spotPython` are:

- Numerical and categorical hyperparameters.
- Powerful surrogate models.
- Flexible approach and easy to use.
- Simple JSON files for the specification of the hyperparameters.
- Extension of default and user specified network classes.
- Noise handling techniques.
- Online visualization of the hyperparameter tuning process with `tensorboard`.

Currently, only rudimentary parallel and distributed neural network training is possible, but these capabilities will be extended in the future. The next version of `spotPython` will also include a more detailed documentation and more examples.

Important

Important: This tutorial does not present a complete benchmarking study (Bartz-Beielstein et al. 2020). The results are only preliminary and highly dependent on the local configuration (hard- and software). Our goal is to provide a first impression of the performance of the hyperparameter tuning package `spotPython`. The results should be interpreted with care.

20 HPT: PyTorch With VBDP

In this tutorial, we will show how `spotPython` can be integrated into the `PyTorch` training workflow for a 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                    0.2.46
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, 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_2023-06-27_04-16-01
```

20.2 Step 2: Initialization of the `fun_control` Dictionary

> **Caution: Tensorboard does not work under Windows**
>
> - Since tensorboard does not work under Windows, we recommend setting the parameter `tensorboard_path` to `None` if you are working under Windows.

`spotPython` uses a Python dictionary for storing the information required for the hyperparameter tuning process, which was described in Section 14.2, see Initialization of the fun_control Dictionary in the documentation.

```
from spotPython.utils.init import fun_control_init
fun_control = fun_control_init(task="classification",
    tensorboard_path="runs/25_spot_torch_vbdp",
    device=DEVICE)
```

20.3 Step 3: PyTorch Data Loading

20.3.1 1. Load VBDP Data

```
import pandas as pd
from sklearn.preprocessing import OrdinalEncoder
train_df = pd.read_csv('./data/VBDP/train.csv')
# remove the id column
train_df = train_df.drop(columns=['id'])
n_samples = train_df.shape[0]
n_features = train_df.shape[1] - 1
target_column = "prognosis"
# # 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

20.3.2 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)
```

```python
# add the dataset to the fun_control
fun_control.update({"data": train_df, # full dataset,
               "train": train,
               "test": test,
               "n_samples": n_samples,
               "target_column": target_column})
```

20.4 Step 4: Specification of the Preprocessing Model

After the training and test data are specified and added to the `fun_control` dictionary, `spotPython` allows the specification of a data preprocessing pipeline, e.g., for the scaling of the data or for the one-hot encoding of categorical variables, see Section 14.4. This feature is not used here, so we do not change the default value (which is `None`).

20.5 Step 5: Select `algorithm` and `core_model_hyper_dict`

20.5.1 Implementing a Configurable Neural Network With spotPython

`spotPython` includes the `Net_vbdp` class which is implemented in the file `netvbdp.py`. The class is imported here.

This class inherits from the class `Net_Core` which is implemented in the file `netcore.py`, see Section 14.5.1.

20.5.2 Add the NN Model to the fun_control Dictionary

```python
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.

```python
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}}
```

20.6 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model`

`spotPython` provides functions for modifying the hyperparameters, their bounds and factors as well as for activating and de-activating hyperparameters without re-compilation of the Python source code. These functions were described in Section 14.6.

> Caution: Small number of epochs for demonstration purposes
>
> - `epochs` and `patience` are set to small values for demonstration purposes. These values are too small for a real application.
> - More resonable values are, e.g.:
>   - `fun_control = modify_hyper_parameter_bounds(fun_control, "epochs", bounds=[7, 9])` and
>   - `fun_control = modify_hyper_parameter_bounds(fun_control, "patience", bounds=[2, 7])`

```
from spotPython.hyperparameters.values import modify_hyper_parameter_bounds

fun_control = modify_hyper_parameter_bounds(fun_control, "_L0", bounds=[n_features, n_feat
fun_control = modify_hyper_parameter_bounds(fun_control, "l1", bounds=[6, 13])
fun_control = modify_hyper_parameter_bounds(fun_control, "epochs", bounds=[2, 3])
fun_control = modify_hyper_parameter_bounds(fun_control, "patience", bounds=[2, 2])
```

```
from spotPython.hyperparameters.values import modify_hyper_parameter_levels
fun_control = modify_hyper_parameter_levels(fun_control, "optimizer",["Adam", "AdamW", "Ad
# fun_control = modify_hyper_parameter_levels(fun_control, "optimizer", ["Adam"])
# fun_control["core_model_hyper_dict"]
```

20.6.1 Optimizers

Optimizers are described in Section 14.6.1.

```
fun_control = modify_hyper_parameter_bounds(fun_control,
    "lr_mult", bounds=[1e-3, 1e-3])
fun_control = modify_hyper_parameter_bounds(fun_control,
    "sgd_momentum", bounds=[0.9, 0.9])
```

20.7 Step 7: Selection of the Objective (Loss) Function

20.7.1 Evaluation

The evaluation procedure requires the specification of two elements:

1. the way how the data is split into a train and a test set (see Section 14.7.1)
2. the loss function (and a metric).

20.7.2 Loss Functions and Metrics

The loss function is specified by the key `"loss_function"`. We will use CrossEntropy loss for the multiclass-classification task.

```
from torch.nn import CrossEntropyLoss
loss_function = CrossEntropyLoss()
fun_control.update({"loss_function": loss_function})
```

20.7.3 Metric

- We will use the MAP@k metric for the evaluation of the model. Here is an example how this metric is calculated.

```
from spotPython.torch.mapk import MAPK
import torch
mapk = MAPK(k=2)
target = torch.tensor([0, 1, 2, 2])
preds = torch.tensor(
    [
        [0.5, 0.2, 0.2],  # 0 is in top 2
        [0.3, 0.4, 0.2],  # 1 is in top 2
        [0.2, 0.4, 0.3],  # 2 is in top 2
        [0.7, 0.2, 0.1],  # 2 isn't in top 2
    ]
)
mapk.update(preds, target)
print(mapk.compute()) # tensor(0.6250)
```

```
tensor(0.6250)
```

```
from spotPython.torch.mapk import MAPK
import torchmetrics
metric_torch = MAPK(k=3)
fun_control.update({"metric_torch": metric_torch})
```

20.8 Step 8: Calling the SPOT Function

20.8.1 Preparing the SPOT Call

The following code passes the information about the parameter ranges and bounds to `spot`.

```
# extract the variable types, names, and bounds
from spotPython.hyperparameters.values import (get_bound_values,
    get_var_name,
    get_var_type,)
var_type = get_var_type(fun_control)
var_name = get_var_name(fun_control)
```

```python
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:

```python
from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control))
```

```
| name         | type   | default   |    lower |    upper | transform             |
|--------------|--------|-----------|----------|----------|-----------------------|
| _L0          | int    | 64        | 6112     | 6112     | None                  |
| l1           | int    | 8         |    6     |   13     | transform_power_2_int |
| dropout_prob | float  | 0.01      |    0     |    0.9   | None                  |
| lr_mult      | float  | 1.0       |    0.001 |    0.001 | None                  |
| batch_size   | int    | 4         |    1     |    4     | transform_power_2_int |
| epochs       | int    | 4         |    2     |    3     | transform_power_2_int |
| k_folds      | int    | 1         |    1     |    1     | None                  |
| patience     | int    | 2         |    2     |    2     | transform_power_2_int |
| optimizer    | factor | SGD       |    0     |    3     | None                  |
| sgd_momentum | float  | 0.0       |    0.9   |    0.9   | None                  |
```

This allows to check if all information is available and if the information is correct.

20.8.2 The Objective Function `fun_torch`

The objective function `fun_torch` is selected next. It implements an interface from `PyTorch`'s training, validation, and testing methods to `spotPython`.

```python
from spotPython.fun.hypertorch import HyperTorch
fun = HyperTorch().fun_torch
```

```python
from spotPython.hyperparameters.values import get_default_hyperparameters_as_array
hyper_dict=TorchHyperDict().load()
X_start = get_default_hyperparameters_as_array(fun_control, hyper_dict)
```

20.8.3 Starting the Hyperparameter Tuning

The spotPython hyperparameter tuning is started by calling the Spot function as described in Section 14.8.4.

```
import numpy as np
from spotPython.spot import spot
from math import inf
spot_tuner = spot.Spot(fun=fun,
                   lower = lower,
                   upper = upper,
                   fun_evals = inf,
                   fun_repeats = 1,
                   max_time = MAX_TIME,
                   noise = False,
                   tolerance_x = np.sqrt(np.spacing(1)),
                   var_type = var_type,
                   var_name = var_name,
                   infill_criterion = "y",
                   n_points = 1,
                   seed=123,
                   log_level = 50,
                   show_models= False,
                   show_progress= True,
                   fun_control = fun_control,
                   design_control={"init_size": INIT_SIZE,
                                   "repeats": 1},
                   surrogate_control={"noise": True,
                                      "cod_type": "norm",
                                      "min_theta": -4,
                                      "max_theta": 3,
                                      "n_theta": len(var_name),
                                      "model_fun_evals": 10_000,
                                      "log_level": 50
                                      })
spot_tuner.run(X_start=X_start)
```

```
config: {'_L0': 6112, 'l1': 2048, 'dropout_prob': 0.17031221661559992, 'lr_mult': 0.001, 'ba
Epoch: 1 |

MAPK: 0.1696428507566452 | Loss: 2.3980519601276944 | Acc: 0.0943396226415094.
```

```
Epoch: 2 | MAPK: 0.1733630895614624 | Loss: 2.3980203526360646 | Acc: 0.0943396226415094.

Epoch: 3 | MAPK: 0.1860118955373764 | Loss: 2.3979072911398753 | Acc: 0.1084905660377359.
Epoch: 4 |

MAPK: 0.1852678507566452 | Loss: 2.3978760242462158 | Acc: 0.1179245283018868.
Epoch: 5 | MAPK: 0.1748512089252472 | Loss: 2.3978382859911238 | Acc: 0.1132075471698113.
Epoch: 6 |

MAPK: 0.1822916418313980 | Loss: 2.3977216482162476 | Acc: 0.1084905660377359.
Epoch: 7 | MAPK: 0.2135416418313980 | Loss: 2.3977128948484148 | Acc: 0.1462264150943396.
Epoch: 8 |

MAPK: 0.2053571194410324 | Loss: 2.3976325137274608 | Acc: 0.1556603773584906.
Returned to Spot: Validation loss: 2.397632513727461

config: {'_L0': 6112, 'l1': 256, 'dropout_prob': 0.19379790035512987, 'lr_mult': 0.001, 'bat
Epoch: 1 |

MAPK: 0.1512345671653748 | Loss: 2.3983973220542625 | Acc: 0.0849056603773585.
Epoch: 2 |

MAPK: 0.1512345671653748 | Loss: 2.3983799528192589 | Acc: 0.0849056603773585.
Epoch: 3 |

MAPK: 0.1512345671653748 | Loss: 2.3983065905394376 | Acc: 0.0849056603773585.
Epoch: 4 |

MAPK: 0.1512345671653748 | Loss: 2.3983387417263455 | Acc: 0.0849056603773585.
Returned to Spot: Validation loss: 2.3983387417263455

config: {'_L0': 6112, 'l1': 4096, 'dropout_prob': 0.6759063718076167, 'lr_mult': 0.001, 'bat
Epoch: 1 |

MAPK: 0.1682389825582504 | Loss: 2.3980356499833881 | Acc: 0.1084905660377359.
Epoch: 2 |

MAPK: 0.1949685811996460 | Loss: 2.3979045669987515 | Acc: 0.1415094339622641.
Epoch: 3 |
```

```
MAPK: 0.1926100701093674 | Loss: 2.3979190003197148 | Acc: 0.1367924528301887.
Epoch: 4 |

MAPK: 0.1784591525793076 | Loss: 2.3977524514468209 | Acc: 0.1132075471698113.
Epoch: 5 |

MAPK: 0.1996855437755585 | Loss: 2.3974908320408947 | Acc: 0.1179245283018868.
Epoch: 6 |

MAPK: 0.1957547217607498 | Loss: 2.3972904457236237 | Acc: 0.1132075471698113.
Epoch: 7 |

MAPK: 0.2059748321771622 | Loss: 2.3968293464408732 | Acc: 0.0943396226415094.
Epoch: 8 |

MAPK: 0.2209119647741318 | Loss: 2.3964822517251068 | Acc: 0.0896226415094340.
Returned to Spot: Validation loss: 2.3964822517251068

config: {'_L0': 6112, 'l1': 128, 'dropout_prob': 0.37306669346546995, 'lr_mult': 0.001, 'batc
Epoch: 1 |

MAPK: 0.1839622408151627 | Loss: 2.3985529260815315 | Acc: 0.1132075471698113.
Epoch: 2 |

MAPK: 0.1839622408151627 | Loss: 2.3984749002276726 | Acc: 0.1132075471698113.
Epoch: 3 |

MAPK: 0.1839622408151627 | Loss: 2.3984067395048321 | Acc: 0.1132075471698113.
Epoch: 4 |

MAPK: 0.1839622408151627 | Loss: 2.3984510313789800 | Acc: 0.1132075471698113.
Returned to Spot: Validation loss: 2.39845103137898

config: {'_L0': 6112, 'l1': 1024, 'dropout_prob': 0.870137281216666, 'lr_mult': 0.001, 'batch
Epoch: 1 |

MAPK: 0.1674382835626602 | Loss: 2.3988769319322376 | Acc: 0.0990566037735849.
Epoch: 2 |
```

```
MAPK: 0.1550925970077515 | Loss: 2.3990008212901928 | Acc: 0.0896226415094340.
Epoch: 3 |

MAPK: 0.1520061641931534 | Loss: 2.3988405863444009 | Acc: 0.0801886792452830.
Epoch: 4 |

MAPK: 0.1628086417913437 | Loss: 2.3990385444075972 | Acc: 0.0849056603773585.
Epoch: 5 |

MAPK: 0.1628086417913437 | Loss: 2.3988307493704335 | Acc: 0.0896226415094340.
Epoch: 6 |

MAPK: 0.1666666716337204 | Loss: 2.3987195226881237 | Acc: 0.1037735849056604.
Epoch: 7 |

MAPK: 0.1612654477357864 | Loss: 2.3986914775989674 | Acc: 0.0990566037735849.
Epoch: 8 |

MAPK: 0.1736111044883728 | Loss: 2.3986782056313976 | Acc: 0.1132075471698113.
Returned to Spot: Validation loss: 2.3986782056313976

config: {'_L0': 6112, 'l1': 4096, 'dropout_prob': 0.678570803516682, 'lr_mult': 0.001, 'batcl
Epoch: 1 |

MAPK: 0.1682390123605728 | Loss: 2.3977888620124674 | Acc: 0.0943396226415094.
Epoch: 2 |

MAPK: 0.1682390421628952 | Loss: 2.3977888485170760 | Acc: 0.0943396226415094.
Epoch: 3 |

MAPK: 0.1666666418313980 | Loss: 2.3975526881667801 | Acc: 0.0896226415094340.
Epoch: 4 |

MAPK: 0.1831761002540588 | Loss: 2.3973054481002518 | Acc: 0.1132075471698113.
Epoch: 5 |

MAPK: 0.1808176040649414 | Loss: 2.3971551746692299 | Acc: 0.1037735849056604.
Epoch: 6 |
```

```
MAPK: 0.1808176189661026 | Loss: 2.3968287121574834 | Acc: 0.1037735849056604.
Epoch: 7 |

MAPK: 0.1871069073677063 | Loss: 2.3964052110348106 | Acc: 0.1037735849056604.
Epoch: 8 |

MAPK: 0.1855345517396927 | Loss: 2.3959594830027164 | Acc: 0.1037735849056604.
Returned to Spot: Validation loss: 2.3959594830027164
spotPython tuning: 2.3959594830027164 [####------] 37.58%


config: {'_L0': 6112, 'l1': 4096, 'dropout_prob': 0.6451395692472426, 'lr_mult': 0.001, 'bat
Epoch: 1 |

MAPK: 0.1965408921241760 | Loss: 2.3977939250334255 | Acc: 0.1320754716981132.
Epoch: 2 |

MAPK: 0.1996855437755585 | Loss: 2.3977641542002841 | Acc: 0.1226415094339623.
Epoch: 3 |

MAPK: 0.1784591525793076 | Loss: 2.3977102401121608 | Acc: 0.0849056603773585.
Epoch: 4 |

MAPK: 0.1902515888214111 | Loss: 2.3976341733392679 | Acc: 0.1132075471698113.
Epoch: 5 |

MAPK: 0.1863207519054413 | Loss: 2.3975428995096459 | Acc: 0.1084905660377359.
Epoch: 6 |

MAPK: 0.1839622408151627 | Loss: 2.3973539437887803 | Acc: 0.0943396226415094.
Epoch: 7 |

MAPK: 0.1800314337015152 | Loss: 2.3972941614546865 | Acc: 0.0943396226415094.
Epoch: 8 |

MAPK: 0.1878931075334549 | Loss: 2.3968496907432124 | Acc: 0.0943396226415094.
Returned to Spot: Validation loss: 2.3968496907432124
spotPython tuning: 2.3959594830027164 [########--] 75.23%
```

```
config: {'_L0': 6112, 'l1': 2048, 'dropout_prob': 0.1703503921641629, 'lr_mult': 0.001, 'bat
Epoch: 1 |

MAPK: 0.1874999850988388 | Loss: 2.3977209159306119 | Acc: 0.1226415094339623.
Epoch: 2 |

MAPK: 0.1815476119518280 | Loss: 2.3977279492786954 | Acc: 0.1273584905660377.
Epoch: 3 |

MAPK: 0.1770833432674408 | Loss: 2.3976931742259433 | Acc: 0.1226415094339623.
Epoch: 4 |

MAPK: 0.1949404627084732 | Loss: 2.3976167951311385 | Acc: 0.1415094339622641.
Epoch: 5 |

MAPK: 0.2120535671710968 | Loss: 2.3975744588034495 | Acc: 0.1509433962264151.
Epoch: 6 |

MAPK: 0.2068452388048172 | Loss: 2.3975220918655396 | Acc: 0.1415094339622641.
Epoch: 7 | MAPK: 0.2313988059759140 | Loss: 2.3974369934626987 | Acc: 0.1603773584905660.
Epoch: 8 |

MAPK: 0.2440476119518280 | Loss: 2.3974144629069736 | Acc: 0.1603773584905660.
Returned to Spot: Validation loss: 2.3974144629069736
spotPython tuning: 2.3959594830027164 [########--] 78.25%


config: {'_L0': 6112, 'l1': 2048, 'dropout_prob': 0.688934449719244, 'lr_mult': 0.001, 'batc
Epoch: 1 |

MAPK: 0.1863207370042801 | Loss: 2.3977806523161114 | Acc: 0.1084905660377359.
Epoch: 2 |

MAPK: 0.1737421154975891 | Loss: 2.3976876398302474 | Acc: 0.0943396226415094.
Epoch: 3 |

MAPK: 0.1918238848447800 | Loss: 2.3976394185480081 | Acc: 0.1084905660377359.
Epoch: 4 |
```

```
MAPK: 0.2099056392908096 | Loss: 2.3974321855688996 | Acc: 0.1320754716981132.
Epoch: 5 |

MAPK: 0.2075471729040146 | Loss: 2.3973877362485201 | Acc: 0.1037735849056604.
Epoch: 6 |

MAPK: 0.2012578547000885 | Loss: 2.3972473032069654 | Acc: 0.1084905660377359.
Epoch: 7 |

MAPK: 0.2051886916160583 | Loss: 2.3973041502934582 | Acc: 0.1084905660377359.
Epoch: 8 |

MAPK: 0.1996855437755585 | Loss: 2.3969096980004942 | Acc: 0.0896226415094340.
Returned to Spot: Validation loss: 2.3969096980004942
spotPython tuning: 2.3959594830027164 [##########] 100.00% Done...
```

```
<spotPython.spot.spot.Spot at 0x29b747220>
```

20.9 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section 14.9, see also the description in the documentation: Tensorboard.

20.10 Step 10: Results

After the hyperparameter tuning run is finished, the results can be analyzed as described in Section 14.10.

```
spot_tuner.plot_progress(log_y=False,
    filename="./figures/" + experiment_name+"_progress.png")
```

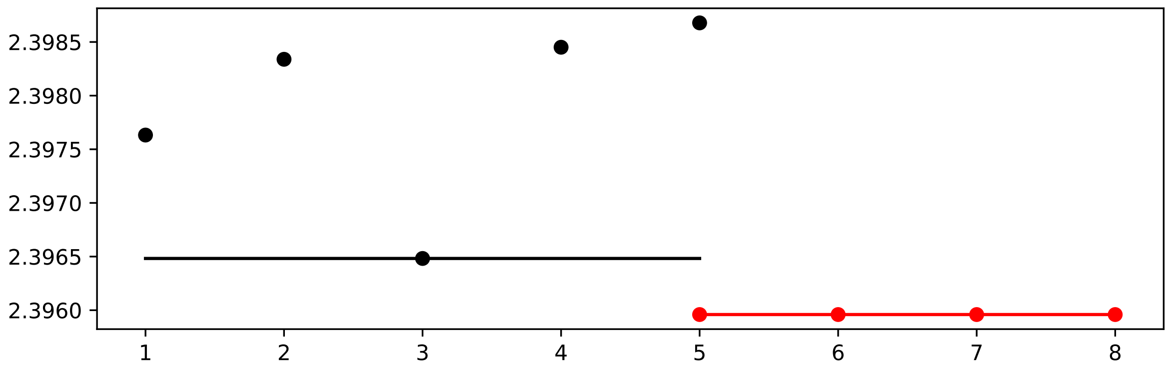


Figure 20.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

```python
from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control=fun_control, spot=spot_tuner))
```

```
| name         | type   | default   |   lower |   upper |               tuned | transform
|--------------|--------|-----------|---------|---------|---------------------|----------------
| _L0          | int    | 64        |  6112.0 |  6112.0 |              6112.0 | None
| l1           | int    | 8         |     6.0 |    13.0 |                12.0 | transform_powe
| dropout_prob | float  | 0.01      |     0.0 |     0.9 | 0.678570803516682   | 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
```

```python
spot_tuner.plot_importance(threshold=0.025,
    filename="./figures/" + experiment_name+"_importance.png")
```

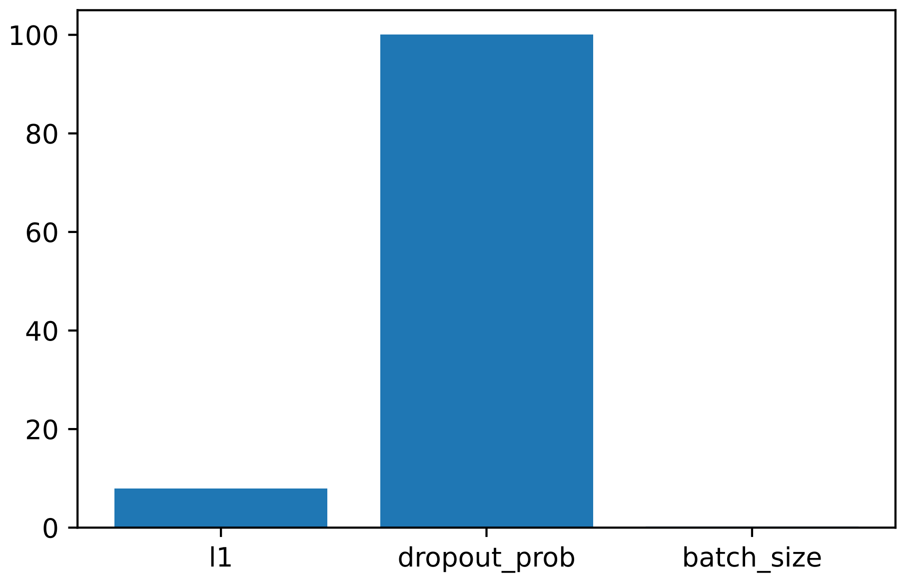


Figure 20.2: Variable importance plot, threshold 0.025.

20.10.1 Get the Tuned Architecture

```
from spotPython.hyperparameters.values import get_one_core_model_from_X
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
model_spot = get_one_core_model_from_X(X, fun_control)
model_spot
```

```
Net_vbdp(
  (fc1): Linear(in_features=6112, out_features=4096, bias=True)
  (fc2): Linear(in_features=4096, out_features=2048, bias=True)
  (fc3): Linear(in_features=2048, out_features=1024, bias=True)
  (fc4): Linear(in_features=1024, out_features=512, bias=True)
  (fc5): Linear(in_features=512, out_features=11, bias=True)
  (relu): ReLU()
  (softmax): Softmax(dim=1)
  (dropout1): Dropout(p=0.678570803516682, inplace=False)
  (dropout2): Dropout(p=0.339285401758341, inplace=False)
)
```

20.10.2 Evaluation of the Tuned Architecture

```
from spotPython.torch.traintest import (
    train_tuned,
    test_tuned,
    )
train_tuned(net=model_spot, train_dataset=train,
        loss_function=fun_control["loss_function"],
        metric=fun_control["metric_torch"],
        shuffle=True,
        device = fun_control["device"],
        path=None,
        task=fun_control["task"],)
```

```
Epoch: 1 |

MAPK: 0.1588050127029419 | Loss: 2.3981116200393102 | Acc: 0.0849056603773585.
Epoch: 2 |

MAPK: 0.1753144711256027 | Loss: 2.3979051383036487 | Acc: 0.1037735849056604.
Epoch: 3 |

MAPK: 0.1729559749364853 | Loss: 2.3978065949565961 | Acc: 0.0801886792452830.
Epoch: 4 |

MAPK: 0.1886792480945587 | Loss: 2.3975860037893617 | Acc: 0.0990566037735849.
Epoch: 5 |

MAPK: 0.1847484111785889 | Loss: 2.3973481880044036 | Acc: 0.0896226415094340.
Epoch: 6 |

MAPK: 0.1540880799293518 | Loss: 2.3972033937022372 | Acc: 0.0660377358490566.
Epoch: 7 |

MAPK: 0.1981132030487061 | Loss: 2.3967299303918517 | Acc: 0.1367924528301887.
Epoch: 8 |

MAPK: 0.1619497090578079 | Loss: 2.3965034259940095 | Acc: 0.0990566037735849.
Returned to Spot: Validation loss: 2.3965034259940095
```

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: 0.1863296180963516 | Loss: 2.3959371481048928 | Acc: 0.0960451977401130.
Final evaluation: Validation loss: 2.3959371481048928
Final evaluation: Validation metric: 0.18632961809635162
----------------------------------------------
```

```
(2.3959371481048928, nan, tensor(0.1863))
```

20.10.3 Cross-validated Evaluations

- This is the evaluation that will be used in the comparison.

> Caution: Cross-validated Evaluations
>
> - The number of folds is set to 1 by default.
> - Here it was changed to 3 for demonstration purposes.
> - Set the number of folds to a reasonable value, e.g., 10.
> - This can be done by setting the `k_folds` attribute of the model as follows:
> - `setattr(model_spot, "k_folds",  10)`

```
from spotPython.torch.traintest import evaluate_cv
# modify k-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: 0.2161016762256622 | Loss: 2.3974435854766329 | Acc: 0.1271186440677966.
Epoch: 2 |

MAPK: 0.2217514067888260 | Loss: 2.3970851837578468 | Acc: 0.1271186440677966.
Epoch: 3 |

MAPK: 0.2210451960563660 | Loss: 2.3967887850131020 | Acc: 0.1398305084745763.
Epoch: 4 |

MAPK: 0.2224575728178024 | Loss: 2.3962903204610790 | Acc: 0.1228813559322034.
Epoch: 5 |

MAPK: 0.2224575877189636 | Loss: 2.3955207052877392 | Acc: 0.1313559322033898.
Epoch: 6 |

MAPK: 0.2295197248458862 | Loss: 2.3947951753260726 | Acc: 0.1271186440677966.
Epoch: 7 |

MAPK: 0.2330508530139923 | Loss: 2.3936606302099714 | Acc: 0.1271186440677966.
Epoch: 8 |

MAPK: 0.2288135290145874 | Loss: 2.3920704570867248 | Acc: 0.1271186440677966.
Fold: 2
Epoch: 1 |

MAPK: 0.2161016464233398 | Loss: 2.3974027269977634 | Acc: 0.1271186440677966.
Epoch: 2 |

MAPK: 0.2344632297754288 | Loss: 2.3971215062222240 | Acc: 0.1271186440677966.
Epoch: 3 |

MAPK: 0.2436439692974091 | Loss: 2.3965833631612488 | Acc: 0.1186440677966102.
Epoch: 4 |

MAPK: 0.2464688718318939 | Loss: 2.3958992028640487 | Acc: 0.1228813559322034.
Epoch: 5 |
```

```
MAPK: 0.2499999403953552 | Loss: 2.3952736187789401 | Acc: 0.1186440677966102.
Epoch: 6 |

MAPK: 0.2365818619728088 | Loss: 2.3940117561211021 | Acc: 0.1186440677966102.
Epoch: 7 |

MAPK: 0.2330508083105087 | Loss: 2.3922617132380859 | Acc: 0.1186440677966102.
Epoch: 8 |

MAPK: 0.2281073182821274 | Loss: 2.3900778697708907 | Acc: 0.1186440677966102.
Fold: 3
Epoch: 1 |

MAPK: 0.2153954803943634 | Loss: 2.3973833261910134 | Acc: 0.1234042553191489.
Epoch: 2 |

MAPK: 0.1857344508171082 | Loss: 2.3970908573118308 | Acc: 0.1063829787234043.
Epoch: 3 |

MAPK: 0.1864406615495682 | Loss: 2.3967134992955095 | Acc: 0.0978723404255319.
Epoch: 4 |

MAPK: 0.1793785393238068 | Loss: 2.3961387832285994 | Acc: 0.0936170212765957.
Epoch: 5 |

MAPK: 0.1737288087606430 | Loss: 2.3953731807611756 | Acc: 0.0893617021276596.
Epoch: 6 |

MAPK: 0.1956214755773544 | Loss: 2.3939166190260548 | Acc: 0.1191489361702128.
Epoch: 7 |

MAPK: 0.1864406764507294 | Loss: 2.3925991300809182 | Acc: 0.1106382978723404.
Epoch: 8 |

MAPK: 0.1970338970422745 | Loss: 2.3903983386896424 | Acc: 0.1276595744680851.
```

```python
metric_name = type(fun_control["metric_torch"]).__name__
print(f"loss: {df_eval}, Cross-validated {metric_name}: {df_metrics}")
```

```
loss: 2.390848888515753, Cross-validated MAPK: 0.21798491477966309
```

20.10.4 Detailed Hyperparameter Plots

```
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

```
l1:  7.946384445740069
dropout_prob:  100.00000000000001
batch_size:  0.20498261417808078
```

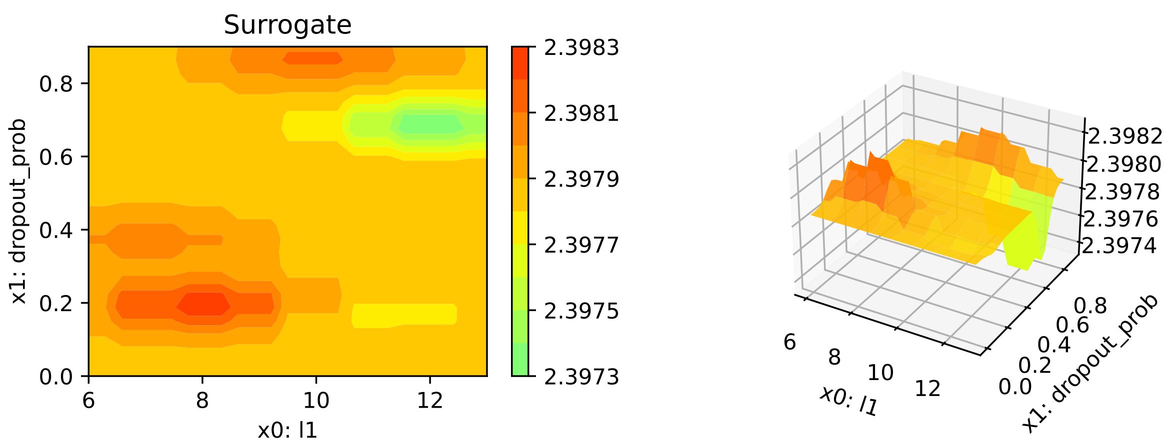
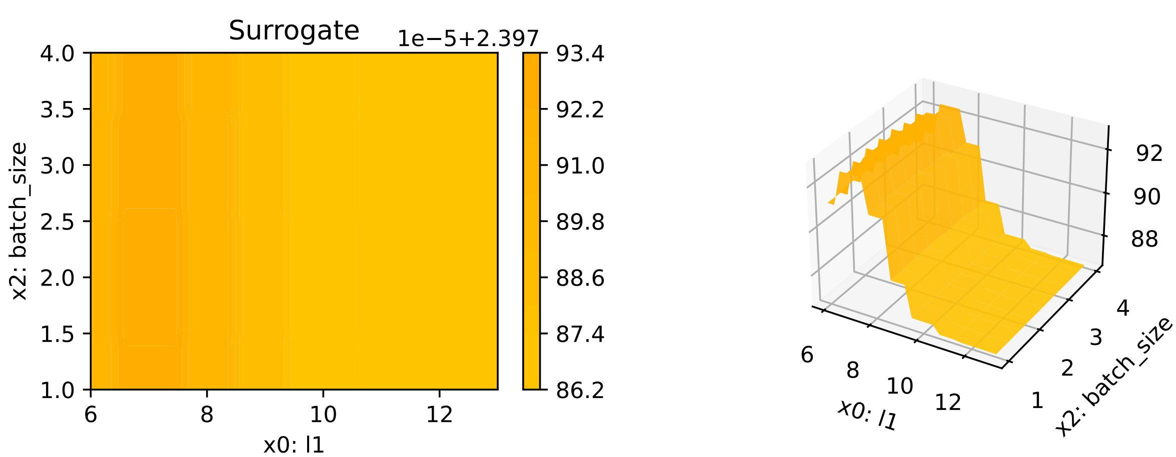
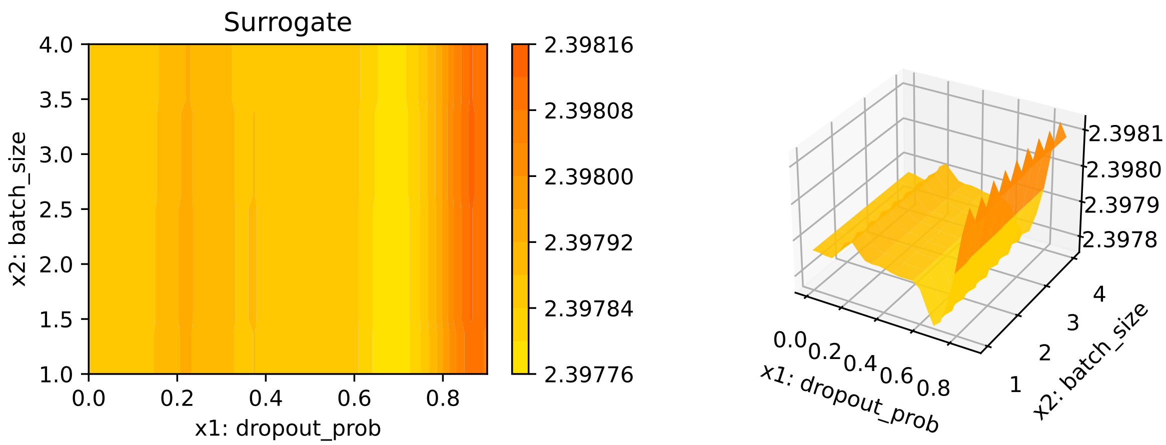


Figure 20.3: Contour plots.





20.10.5 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

```
Unable to display output for mime type(s): text/html
```

Parallel coordinates plots

```
Unable to display output for mime type(s): text/html
```

```
# close tensorbaoard writer
if fun_control["writer"] is not None:
    fun_control["writer"].close()
```

20.10.6 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

```
PLOT_ALL = False
if PLOT_ALL:
    n = spot_tuner.k
    for i in range(n-1):
        for j in range(i+1, n):
            spot_tuner.plot_contour(i=i, j=j, min_z=min_z, max_z = max_z)
```

21 HPT PyTorch Lightning: VBDP

In this tutorial, we will show how `spotPython` can be integrated into the `PyTorch` Lightning training workflow for a classifiaction task.

> Caution: Data must be downloaded manually
>
> - Ensure that the corresponding data is available as `./data/VBDP/train.csv`.

This document refers to the following software versions:

- `python`: 3.10.10
- `torch`: 2.0.1
- `torchvision`: 0.15.0

```
pip list | grep  "spot[RiverPython]"
```

```
spotPython                    0.2.46
```

```
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, initial design size and the device that is used.

Caution: Run time and initial design size should be increased for real experiments

- MAX_TIME is set to one minute for demonstration purposes. For real experiments, this should be increased to at least 1 hour.
- INIT_SIZE is set to 5 for demonstration purposes. For real experiments, this should be increased to at least 10.
- WORKERS is set to 0 for demonstration purposes. For real experiments, this should be increased. See the warnings that are printed when the number of workers is set to 0.

Note: Device selection

- The device can be selected by setting the variable `DEVICE`.
- Since we are using a simple neural net, the setting `"cpu"` is preferred (on Mac).
- If you have a GPU, you can use `"cuda:0"` instead.
- If DEVICE is set to `"auto"` or `None`, `spotPython` will automatically select the device.
  - This might result in `"mps"` on Macs, which is not the best choice for simple neural nets.

Note: Prefix

- The prefix `PREFIX` is used for the experiment name and the name of the log file.

```
MAX_TIME = 1
INIT_SIZE = 5
DEVICE = "auto" #"cpu" # "cuda:0"
WORKERS = 0
PREFIX="30"
```

```
from spotPython.utils.device import getDevice
DEVICE = getDevice(DEVICE)
print(DEVICE)
```

```
mps
```

```python
import os
if not os.path.exists('./figures'):
    os.makedirs('./figures')
```

21.2 Step 2: Initialization of the `fun_control` Dictionary

> Caution: Tensorboard does not work under Windows
>
> - Since tensorboard does not work under Windows, we recommend setting the parameter `tensorboard_path` to `None` if you are working under Windows.

`spotPython` uses a Python dictionary for storing the information required for the hyperparameter tuning process, which was described in Section 14.2, see Initialization of the fun_control Dictionary in the documentation.

```python
from spotPython.utils.init import fun_control_init
from spotPython.utils.file import get_experiment_name
experiment_name = get_experiment_name(prefix=PREFIX)
fun_control = fun_control_init(task="classification",
    tensorboard_path="./runs/" + experiment_name,
    num_workers=WORKERS,
    device=DEVICE)
```

21.3 Step 3: PyTorch Data Loading

21.3.1 Lightning Dataset and DataModule

The data loading and preprocessing is handled by `Lightning` and `PyTorch`. It comprehends the following classes:

- `CSVDataset`: A class that loads the data from a CSV file. [SOURCE]
- `CSVDataModule`: A class that prepares the data for training and testing. [SOURCE]

21.3.1.1 Taking a Look at the Data

```
import torch
from spotPython.light.csvdataset import CSVDataset
from torch.utils.data import DataLoader
from torchvision.transforms import ToTensor

# Create an instance of CSVDataset
dataset = CSVDataset(csv_file="./data/VBDP/train.csv", train=True)
# show the dimensions of the input data
print(dataset[0][0].shape)
# show the first element of the input data
print(dataset[0][0])
# show the size of the dataset
print(f"Dataset Size: {len(dataset)}")
```

```
torch.Size([64])
tensor([1., 1., 0., 1., 1., 1., 1., 0., 1., 1., 1., 1., 0., 0., 1., 1., 0., 0.,
        1., 0., 1., 0., 1., 1., 1., 1., 1., 1., 1., 0., 0., 1., 0., 0., 0., 0.,
        1., 0., 0., 0., 0., 0., 1., 0., 1., 0., 1., 0., 0., 0., 0., 1., 0., 1.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
Dataset Size: 707
```

```
# Set batch size for DataLoader
batch_size = 3
# Create DataLoader
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)

# Iterate over the data in the DataLoader
for batch in dataloader:
    inputs, targets = batch
    print(f"Batch Size: {inputs.size(0)}")
    print("---------------")
    print(f"Inputs: {inputs}")
    print(f"Targets: {targets}")
    break
```

```
Batch Size: 3
---------------
Inputs: tensor([[1., 0., 1., 0., 1., 0., 0., 1., 1., 1., 0., 1., 0., 0., 1., 1., 1., 0.,
```

```
         1., 0., 0., 0., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 1., 0., 1., 1.,
         1., 1., 1., 1., 1., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [1., 1., 0., 1., 1., 1., 1., 1., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1.,
         1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0.,
         1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 0.,
         0., 0., 1., 0., 0., 1., 1., 1., 0., 0.]])
Targets: tensor([10,  5,  7])
```

> Caution: Data Loading in Lightning
>
> - Data loading is handled independently from the `fun_control` dictionary by `Lightning` and `PyTorch`.
> - In contrast to `spotPython` with `torch`, `river` and `sklearn`, the data sets are not added to the `fun_control` dictionary.

21.4 Step 4: Specification of the Preprocessing Model

The `fun_control` dictionary, the `torch`, `sklearn`and `river` versions of `spotPython` allow the specification of a data preprocessing pipeline, e.g., for the scaling of the data or for the one-hot encoding of categorical variables, see Section 14.4. This feature is not used in the `Lightning` version.

> Caution: Data preprocessing in Lightning
>
> Lightning allows the data preprocessing to be specified in the `LightningDataModule` class. It is not considered here, because it should be computed at one location only.

21.5 Step 5: Select the NN Model (`algorithm`) and `core_model_hyper_dict`

21.5.1 Implementing a Configurable Neural Network With spotPython

`spotPython` includes the `NetLightBase` class [SOURCE] for configurable neural networks. The class is imported here. It inherits from the class `Lightning.LightningModule`, which is the base class for all models in `Lightning`. `Lightning.LightningModule` is a subclass of `torch.nn.Module` and provides additional functionality for the training and testing of neural networks. The class `Lightning.LightningModule` is described in the Lightning documentation.

21.5.2 Add the NN Model to the fun_control Dictionary

```
from spotPython.light.netlightbase import NetLightBase
from spotPython.data.light_hyper_dict import LightHyperDict
from spotPython.hyperparameters.values import add_core_model_to_fun_control
fun_control = add_core_model_to_fun_control(core_model=NetLightBase,
                              fun_control=fun_control,
                              hyper_dict= LightHyperDict)
```

The default entries for the `core_model` class are shown below.

```
fun_control['core_model_hyper_dict']
```

```
{'l1': {'type': 'int',
  'default': 3,
  'transform': 'transform_power_2_int',
  'lower': 3,
  'upper': 8},
 'epochs': {'type': 'int',
  'default': 4,
  'transform': 'transform_power_2_int',
  'lower': 4,
  'upper': 9},
 'batch_size': {'type': 'int',
  'default': 4,
  'transform': 'transform_power_2_int',
  'lower': 1,
  'upper': 4},
 'act_fn': {'levels': ['Sigmoid', 'Tanh', 'ReLU', 'LeakyReLU', 'ELU', 'Swish'],
  'type': 'factor',
  'default': 'ReLU',
  'transform': 'None',
  'class_name': 'spotPython.torch.activation',
  'core_model_parameter_type': 'instance()',
  'lower': 0,
  'upper': 2},
```

```
'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': 11},
 'dropout_prob': {'type': 'float',
  'default': 0.01,
  'transform': 'None',
  'lower': 0.0,
  'upper': 0.1},
 'lr_mult': {'type': 'float',
  'default': 1.0,
  'transform': 'None',
  'lower': 0.1,
  'upper': 10.0}}
```

The `NetLightBase` is a configurable neural network. The hyperparameters of the model are specified in the `core_model_hyper_dict` dictionary [SOURCE].

21.6 Step 6: Modify `hyper_dict` Hyperparameters for the Selected Algorithm aka `core_model`

`spotPython` provides functions for modifying the hyperparameters, their bounds and factors as well as for activating and de-activating hyperparameters without re-compilation of the Python source code. These functions were described in Section 14.6.

Caution: Small number of epochs for demonstration purposes

- `epochs` and `patience` are set to small values for demonstration purposes. These values are too small for a real application.
- More resonable values are, e.g.:
  - `fun_control = modify_hyper_parameter_bounds(fun_control, "epochs", bounds=[7, 9])` and
  - `fun_control = modify_hyper_parameter_bounds(fun_control, "patience", bounds=[2, 7])`

```
from spotPython.hyperparameters.values import modify_hyper_parameter_bounds

fun_control = modify_hyper_parameter_bounds(fun_control, "l1", bounds=[2,3])
fun_control = modify_hyper_parameter_bounds(fun_control, "epochs", bounds=[1,2])
fun_control = modify_hyper_parameter_bounds(fun_control, "batch_size", bounds=[6, 8])
```

```
from spotPython.hyperparameters.values import modify_hyper_parameter_levels
fun_control = modify_hyper_parameter_levels(fun_control, "optimizer",["Adam", "AdamW", "Ad
# fun_control = modify_hyper_parameter_levels(fun_control, "optimizer", ["Adam"])
```

The updated `fun_control` dictionary is shown below.

```
fun_control["core_model_hyper_dict"]
```

```
{'l1': {'type': 'int',
  'default': 3,
  'transform': 'transform_power_2_int',
  'lower': 2,
  'upper': 3},
 'epochs': {'type': 'int',
  'default': 4,
  'transform': 'transform_power_2_int',
  'lower': 1,
  'upper': 2},
 'batch_size': {'type': 'int',
  'default': 4,
  'transform': 'transform_power_2_int',
  'lower': 6,
  'upper': 8},
 'act_fn': {'levels': ['Sigmoid', 'Tanh', 'ReLU', 'LeakyReLU', 'ELU', 'Swish'],
```

```
 'type': 'factor',
 'default': 'ReLU',
 'transform': 'None',
 'class_name': 'spotPython.torch.activation',
 'core_model_parameter_type': 'instance()',
 'lower': 0,
 'upper': 2},
'optimizer': {'levels': ['Adam', 'AdamW', 'Adamax', 'NAdam'],
 'type': 'factor',
 'default': 'SGD',
 'transform': 'None',
 'class_name': 'torch.optim',
 'core_model_parameter_type': 'str',
 'lower': 0,
 'upper': 3},
'dropout_prob': {'type': 'float',
 'default': 0.01,
 'transform': 'None',
 'lower': 0.0,
 'upper': 0.1},
'lr_mult': {'type': 'float',
 'default': 1.0,
 'transform': 'None',
 'lower': 0.1,
 'upper': 10.0}}
```

21.7 Step 7: Data Splitting, the Objective (Loss) Function and the Metric

21.7.1 Evaluation

The evaluation procedure requires the specification of two elements:

1. the way how the data is split into a train and a test set (see Section 14.7.1)
2. the loss function (and a metric).

> Caution: Data Splitting in Lightning
>
> - The data splitting is handled by `Lightning`.

21.7.2 Loss Functions and Metrics

The loss function is specified in the configurable network class [SOURCE] We will use CrossEntropy loss for the multiclass-classification task.

21.7.3 Metric

- We will use the MAP@k metric [SOURCE] for the evaluation of the model. Here is an example how this metric is calculated.

```python
from spotPython.torch.mapk import MAPK
import torch
mapk = MAPK(k=2)
target = torch.tensor([0, 1, 2, 2])
preds = torch.tensor(
    [
        [0.5, 0.2, 0.2],  # 0 is in top 2
        [0.3, 0.4, 0.2],  # 1 is in top 2
        [0.2, 0.4, 0.3],  # 2 is in top 2
        [0.7, 0.2, 0.1],  # 2 isn't in top 2
    ]
)
mapk.update(preds, target)
print(mapk.compute()) # tensor(0.6250)
```

```
tensor(0.6250)
```

Similar to the loss function, the metric is specified in the configurable network class [SOURCE].

> **Caution: Loss Function and Metric in Lightning**
>
> - The loss function and the metric are not hyperparameters that can be tuned with `spotPython`.
> - They are handled by `Lightning`.

21.8 Step 8: Calling the SPOT Function

21.8.1 Preparing the SPOT Call

The following code passes the information about the parameter ranges and bounds to `spot`. It extracts the variable types, names, and bounds

```
from spotPython.hyperparameters.values import (get_bound_values,
    get_var_name,
    get_var_type,)
var_type = get_var_type(fun_control)
var_name = get_var_name(fun_control)
fun_control.update({"var_type": var_type,
                    "var_name": var_name})
lower = get_bound_values(fun_control, "lower")
upper = get_bound_values(fun_control, "upper")
```

Now, the dictionary `fun_control` contains all information needed for the hyperparameter tuning. Before the hyperparameter tuning is started, it is recommended to take a look at the experimental design. The method `gen_design_table` [SOURCE] generates a design table as follows:

```
from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control))
```

```
| name         | type   | default   |   lower |   upper | transform              |
|--------------|--------|-----------|---------|---------|------------------------|
| l1           | int    | 3         |     2   |     3   | transform_power_2_int |
| epochs       | int    | 4         |     1   |     2   | transform_power_2_int |
| batch_size   | int    | 4         |     6   |     8   | transform_power_2_int |
| act_fn       | factor | ReLU      |     0   |     2   | None                   |
| optimizer    | factor | SGD       |     0   |     3   | None                   |
| dropout_prob | float  | 0.01      |     0   |     0.1 | None                   |
| lr_mult      | float  | 1.0       |     0.1 |    10   | None                   |
```

This allows to check if all information is available and if the information is correct.

21.8.2 The Objective Function `fun`

The objective function `fun` from the class `HyperLight` [SOURCE] is selected next. It implements an interface from `PyTorch`'s training, validation, and testing methods to `spotPython`.

```
from spotPython.light.hyperlight import HyperLight
fun = HyperLight().fun
```

21.8.3 Starting the Hyperparameter Tuning

The `spotPython` hyperparameter tuning is started by calling the `Spot` function [SOURCE] as described in Section 14.8.4.

```
import numpy as np
from spotPython.spot import spot
from math import inf
spot_tuner = spot.Spot(fun=fun,
                   lower = lower,
                   upper = upper,
                   fun_evals = inf,
                   fun_repeats = 1,
                   max_time = MAX_TIME,
                   noise = False,
                   tolerance_x = np.sqrt(np.spacing(1)),
                   var_type = var_type,
                   var_name = var_name,
                   infill_criterion = "y",
                   n_points = 1,
                   seed=123,
                   log_level = 50,
                   show_models= False,
                   show_progress= True,
                   fun_control = fun_control,
                   design_control={"init_size": INIT_SIZE,
                                   "repeats": 1},
                   surrogate_control={"noise": True,
                                      "cod_type": "norm",
                                      "min_theta": -4,
                                      "max_theta": 3,
                                      "n_theta": len(var_name),
                                      "model_fun_evals": 10_000,
                                      "log_level": 50
                                      })
spot_tuner.run()
```

```
config: {'l1': 4, 'epochs': 2, 'batch_size': 256, 'act_fn': Tanh(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=4, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.02375810986688453, inplace=False)
    (3): Linear(in_features=4, out_features=2, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.02375810986688453, inplace=False)
    (6): Linear(in_features=2, out_features=2, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.02375810986688453, inplace=False)
    (9): Linear(in_features=2, out_features=1, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.02375810986688453, inplace=False)
    (12): Linear(in_features=1, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

Validate metric	DataLoader 0
val_acc	0.09187278896570206
val_loss	2.4026222229003906
valid_mapk	0.15528549253940582

```
train_model result: {'valid_mapk': 0.15528549253940582, 'val_loss': 2.4026222229003906, 'val_

config: {'l1': 8, 'epochs': 2, 'batch_size': 128, 'act_fn': Sigmoid(), 'optimizer': 'AdamW',
model: NetLightBase(
  (act_fn): Sigmoid()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Sigmoid()
    (2): Dropout(p=0.06351516807805178, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Sigmoid()
    (5): Dropout(p=0.06351516807805178, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Sigmoid()
    (8): Dropout(p=0.06351516807805178, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Sigmoid()
    (11): Dropout(p=0.06351516807805178, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

Validate metric	DataLoader 0
val_acc	0.06713780760765076
val_loss	2.404522657394409
valid_mapk	0.1134580671787262

```
train_model result: {'valid_mapk': 0.1134580671787262, 'val_loss': 2.404522657394409, 'val_ac

config: {'l1': 8, 'epochs': 4, 'batch_size': 64, 'act_fn': ReLU(), 'optimizer': 'NAdam', 'dro
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.009636699262945718, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.009636699262945718, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.009636699262945718, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.009636699262945718, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

```
Validate metric           DataLoader 0

    val_acc           0.11307420581579208
   val_loss            2.39654541015625
  valid_mapk          0.17509645223617554


train_model result: {'valid_mapk': 0.17509645223617554, 'val_loss': 2.39654541015625, 'val_ac

config: {'l1': 8, 'epochs': 2, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'AdamW', 'di
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.0905773354221908, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.0905773354221908, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.0905773354221908, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.0905773354221908, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]


Training: 0it [00:00, ?it/s]


Validation: 0it [00:00, ?it/s]


Validation: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc              0.08833922445774078
         val_loss              2.4006454944610596
        valid_mapk             0.15690907835960388


train_model result: {'valid_mapk': 0.15690907835960388, 'val_loss': 2.4006454944610596, 'val_

config: {'l1': 4, 'epochs': 4, 'batch_size': 128, 'act_fn': Sigmoid(), 'optimizer': 'Adamax'
model: NetLightBase(
  (act_fn): Sigmoid()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=4, bias=True)
    (1): Sigmoid()
    (2): Dropout(p=0.04267745138523515, inplace=False)
    (3): Linear(in_features=4, out_features=2, bias=True)
    (4): Sigmoid()
    (5): Dropout(p=0.04267745138523515, inplace=False)
    (6): Linear(in_features=2, out_features=2, bias=True)
    (7): Sigmoid()
    (8): Dropout(p=0.04267745138523515, inplace=False)
    (9): Linear(in_features=2, out_features=1, bias=True)
    (10): Sigmoid()
    (11): Dropout(p=0.04267745138523515, inplace=False)
    (12): Linear(in_features=1, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc              0.0989399328827858
         val_loss               2.399477481842041
        valid_mapk             0.15409593284130096


train_model result: {'valid_mapk': 0.15409593284130096, 'val_loss': 2.399477481842041, 'val_a

config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax', 'c
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.03997734726269851, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.03997734726269851, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.03997734726269851, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.03997734726269851, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)
```

```
Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric              DataLoader 0

          val_acc              0.10600706934928894
         val_loss               2.394681692123413
        valid_mapk             0.21291474997997284


train_model result: {'valid_mapk': 0.21291474997997284, 'val_loss': 2.394681692123413, 'val_a
spotPython tuning: 2.394681692123413 [----------] 1.00%


config: {'l1': 8, 'epochs': 4, 'batch_size': 256, 'act_fn': ReLU(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.06565812937559068, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.06565812937559068, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
```

```
    (7): ReLU()
    (8): Dropout(p=0.06565812937559068, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.06565812937559068, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric            DataLoader 0

          val_acc             0.09540636092424393
         val_loss             2.4005861282348633
        valid_mapk            0.14943817257881165


train_model result: {'valid_mapk': 0.14943817257881165, 'val_loss': 2.4005861282348633, 'val_
spotPython tuning: 2.394681692123413 [----------] 1.82%


config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'AdamW', 'd
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
```

```
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.02848230824661844, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.02848230824661844, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.02848230824661844, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.02848230824661844, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

Validate metric	DataLoader 0
val_acc	0.08127208799123764
val_loss	2.4030637741088867
valid_mapk	0.12223508208990097

```
train_model result: {'valid_mapk': 0.12223508208990097, 'val_loss': 2.4030637741088867, 'val_
spotPython tuning: 2.394681692123413 [----------] 2.62%


config: {'l1': 8, 'epochs': 4, 'batch_size': 64, 'act_fn': ReLU(), 'optimizer': 'NAdam', 'dro
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.07042775010542907, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.07042775010542907, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.07042775010542907, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.07042775010542907, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]


Training: 0it [00:00, ?it/s]


Validation: 0it [00:00, ?it/s]


Validation: 0it [00:00, ?it/s]


Validation: 0it [00:00, ?it/s]


Validation: 0it [00:00, ?it/s]


Validation: 0it [00:00, ?it/s]
```

```
      Validate metric            DataLoader 0

          val_acc              0.09187278896570206
         val_loss              2.3988873958587646
        valid_mapk             0.17542438209056854
```

```
train_model result: {'valid_mapk': 0.17542438209056854, 'val_loss': 2.3988873958587646, 'val_
spotPython tuning: 2.394681692123413 [----------] 3.65%
```

```
config: {'l1': 8, 'epochs': 2, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.06913066038307081, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.06913066038307081, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.06913066038307081, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.06913066038307081, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)
```

```
Sanity Checking: 0it [00:00, ?it/s]
```

```
Training: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc              0.08833922445774078
         val_loss               2.396458148956299
        valid_mapk             0.17611880600452423


train_model result: {'valid_mapk': 0.17611880600452423, 'val_loss': 2.396458148956299, 'val_a
spotPython tuning: 2.394681692123413 [----------] 4.35%


config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.07187827116844328, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.07187827116844328, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.07187827116844328, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.07187827116844328, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]
```

```
Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

Validate metric	DataLoader 0
val_acc	0.07773851603269577
val_loss	2.3956966400146484
valid_mapk	0.14660494029521942

```
train_model result: {'valid_mapk': 0.14660494029521942, 'val_loss': 2.3956966400146484, 'val_
spotPython tuning: 2.394681692123413 [#---------] 5.33%
```

```
config: {'l1': 4, 'epochs': 2, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=4, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.04894013498641, inplace=False)
    (3): Linear(in_features=4, out_features=2, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.04894013498641, inplace=False)
    (6): Linear(in_features=2, out_features=2, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.04894013498641, inplace=False)
```

```
    (9): Linear(in_features=2, out_features=1, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.04894013498641, inplace=False)
    (12): Linear(in_features=1, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc             0.07067137956619263
         val_loss             2.4021072387695312
        valid_mapk            0.12797389924526215


train_model result: {'valid_mapk': 0.12797389924526215, 'val_loss': 2.4021072387695312, 'val_
spotPython tuning: 2.394681692123413 [#---------] 6.14%


config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.04716573794255629, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
```

```
    (4): ReLU()
    (5): Dropout(p=0.04716573794255629, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.04716573794255629, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.04716573794255629, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc             0.07067137956619263
         val_loss              2.401764392852783
        valid_mapk            0.1458333283662796


train_model result: {'valid_mapk': 0.1458333283662796, 'val_loss': 2.401764392852783, 'val_a
spotPython tuning: 2.394681692123413 [#---------] 7.10%
```

```
config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax', 'c
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.039134562381799104, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.039134562381799104, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.039134562381799104, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.039134562381799104, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

```
Validate metric           DataLoader 0

    val_acc             0.08833922445774078
   val_loss             2.3973543643951416
  valid_mapk            0.18833591043949127


train_model result: {'valid_mapk': 0.18833591043949127, 'val_loss': 2.3973543643951416, 'val_
spotPython tuning: 2.394681692123413 [#---------] 8.11%


config: {'l1': 4, 'epochs': 4, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=4, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.0720891034114133, inplace=False)
    (3): Linear(in_features=4, out_features=2, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.0720891034114133, inplace=False)
    (6): Linear(in_features=2, out_features=2, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.0720891034114133, inplace=False)
    (9): Linear(in_features=2, out_features=1, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.0720891034114133, inplace=False)
    (12): Linear(in_features=1, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric            DataLoader 0

          val_acc             0.09540636092424393
         val_loss              2.397514820098877
        valid_mapk            0.14924125373363495


train_model result: {'valid_mapk': 0.14924125373363495, 'val_loss': 2.397514820098877, 'val_a
spotPython tuning: 2.394681692123413 [#---------] 9.08%


config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax', '
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.07122752865764566, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.07122752865764566, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.07122752865764566, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.07122752865764566, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)
```

```
Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

Validate metric	DataLoader 0
val_acc	0.12720848619937897
val_loss	2.3984813690185547
valid_mapk	0.1834651380777359

```
train_model result: {'valid_mapk': 0.1834651380777359, 'val_loss': 2.3984813690185547, 'val_a
spotPython tuning: 2.394681692123413 [#---------] 9.98%


config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.07111195274301181, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.07111195274301181, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
```

```
    (7): ReLU()
    (8): Dropout(p=0.07111195274301181, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.07111195274301181, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc             0.10247349739074707
         val_loss              2.397064208984375
        valid_mapk            0.19320665299892426


train_model result: {'valid_mapk': 0.19320665299892426, 'val_loss': 2.397064208984375, 'val_a
spotPython tuning: 2.394681692123413 [#---------] 10.94%


config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
```

```
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.0710633812209252, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.0710633812209252, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.0710633812209252, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.0710633812209252, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

Validate metric	DataLoader 0
val_acc	0.11660777032375336
val_loss	2.397106885910034
valid_mapk	0.19269226491451263

```
train_model result: {'valid_mapk': 0.19269226491451263, 'val_loss': 2.397106885910034, 'val_a

spotPython tuning: 2.394681692123413 [#---------] 12.63%


config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.07095483348264997, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.07095483348264997, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.07095483348264997, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.07095483348264997, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc              0.13780918717384338
         val_loss                2.3945152759552
        valid_mapk             0.21579217910766602


train_model result: {'valid_mapk': 0.21579217910766602, 'val_loss': 2.3945152759552, 'val_ac

spotPython tuning: 2.3945152759552 [#---------] 14.17%


config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax', '
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.07078395789666073, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.07078395789666073, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.07078395789666073, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.07078395789666073, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc             0.13074204325675964
         val_loss              2.391833543777466
        valid_mapk            0.21686922013759613


train_model result: {'valid_mapk': 0.21686922013759613, 'val_loss': 2.391833543777466, 'val_a

spotPython tuning: 2.391833543777466 [##--------] 15.48%


config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.0705497236228775, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.0705497236228775, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.0705497236228775, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
```

```
    (10): ReLU()
    (11): Dropout(p=0.0705497236228775, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc              0.06713780760765076
         val_loss               2.398699998855591
        valid_mapk             0.1760866791009903


train_model result: {'valid_mapk': 0.1760866791009903, 'val_loss': 2.398699998855591, 'val_ac

spotPython tuning: 2.391833543777466 [##--------] 16.99%


config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax', 'c
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
```

```
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.07088501976050601, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.07088501976050601, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.07088501976050601, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.07088501976050601, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc             0.11307420581579208
         val_loss             2.3927626609802246
        valid_mapk            0.19410687685012817


train_model result: {'valid_mapk': 0.19410687685012817, 'val_loss': 2.3927626609802246, 'val
```

```
spotPython tuning: 2.391833543777466 [##--------] 18.25%


config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.07077905401817546, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.07077905401817546, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.07077905401817546, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.07077905401817546, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]


Training: 0it [00:00, ?it/s]


Validation: 0it [00:00, ?it/s]


Validation: 0it [00:00, ?it/s]


Validation: 0it [00:00, ?it/s]


Validation: 0it [00:00, ?it/s]


Validation: 0it [00:00, ?it/s]
```

```
Validate metric             DataLoader 0

    val_acc              0.08127208799123764
    val_loss              2.399000644683838
   valid_mapk            0.12601272761821747


train_model result: {'valid_mapk': 0.12601272761821747, 'val_loss': 2.399000644683838, 'val_a

spotPython tuning: 2.391833543777466 [##--------] 19.60%


config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.07093933583018028, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.07093933583018028, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.07093933583018028, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.07093933583018028, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]


Training: 0it [00:00, ?it/s]


Validation: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc              0.08480565249919891
         val_loss               2.400378465652466
        valid_mapk             0.1524723619222641


train_model result: {'valid_mapk': 0.1524723619222641, 'val_loss': 2.400378465652466, 'val_ac

spotPython tuning: 2.391833543777466 [##--------] 20.94%


config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax', 'c
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.07105639815979657, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.07105639815979657, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.07105639815979657, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.07105639815979657, inplace=False)
```

```
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc              0.208480566740036
         val_loss              2.391202926635742
        valid_mapk            0.25774821639060974


train_model result: {'valid_mapk': 0.25774821639060974, 'val_loss': 2.391202926635742, 'val_a

spotPython tuning: 2.391202926635742 [##--------] 22.48%


config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
```

```
    (1): ReLU()
    (2): Dropout(p=0.07112784401926231, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.07112784401926231, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.07112784401926231, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.07112784401926231, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric              DataLoader 0

          val_acc             0.10954063385725021
         val_loss              2.394171953201294
        valid_mapk            0.17645639181137085


train_model result: {'valid_mapk': 0.17645639181137085, 'val_loss': 2.394171953201294, 'val_a

spotPython tuning: 2.391202926635742 [##--------] 23.93%
```

```
config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax', 'c
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.07115071050032068, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.07115071050032068, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.07115071050032068, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.07115071050032068, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

```
Validate metric           DataLoader 0

    val_acc            0.08833922445774078
   val_loss            2.3992669582366943
  valid_mapk           0.1543370634317398
```

```
train_model result: {'valid_mapk': 0.1543370634317398, 'val_loss': 2.3992669582366943, 'val_a

spotPython tuning: 2.391202926635742 [###-------] 25.20%
```

```
config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.07110916717423099, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.07110916717423099, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.07110916717423099, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.07110916717423099, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc              0.0742049440741539
         val_loss              2.4012465476989746
        valid_mapk             0.1263502985239029


train_model result: {'valid_mapk': 0.1263502985239029, 'val_loss': 2.4012465476989746, 'val_a

spotPython tuning: 2.391202926635742 [###-------] 26.54%


config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.0710801307392398, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.0710801307392398, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.0710801307392398, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.0710801307392398, inplace=False)
```

```
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

Validate metric	DataLoader 0
val_acc	0.0989399328827858
val_loss	2.3983724117279053
valid_mapk	0.17716370522975922

```
train_model result: {'valid_mapk': 0.17716370522975922, 'val_loss': 2.3983724117279053, 'val_

spotPython tuning: 2.391202926635742 [###-------] 27.92%

config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
```

```
    (1): ReLU()
    (2): Dropout(p=0.07107541766227407, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.07107541766227407, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.07107541766227407, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.07107541766227407, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric            DataLoader 0

          val_acc             0.04593639448285103
         val_loss             2.4031989574432373
        valid_mapk            0.1052919253706932


train_model result: {'valid_mapk': 0.1052919253706932, 'val_loss': 2.4031989574432373, 'val_a

spotPython tuning: 2.391202926635742 [###-------] 29.22%
```

```
config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'Adamax', 'c
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.07124731324309404, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.07124731324309404, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.07124731324309404, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.07124731324309404, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

```
     Validate metric           DataLoader 0

         val_acc             0.07067137956619263
        val_loss             2.3995754718780518
       valid_mapk            0.15419238805770874
```

```
train_model result: {'valid_mapk': 0.15419238805770874, 'val_loss': 2.3995754718780518, 'val_

spotPython tuning: 2.391202926635742 [###-------] 30.58%


config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': Tanh(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.06698990155084411, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.06698990155084411, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.06698990155084411, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.06698990155084411, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc              0.07773851603269577
         val_loss               2.393343687057495
        valid_mapk             0.19050604104995728


train_model result: {'valid_mapk': 0.19050604104995728, 'val_loss': 2.393343687057495, 'val_a

spotPython tuning: 2.391202926635742 [###-------] 32.00%


config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': Sigmoid(), 'optimizer': 'Adamax'
model: NetLightBase(
  (act_fn): Sigmoid()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Sigmoid()
    (2): Dropout(p=0.09197568972148612, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Sigmoid()
    (5): Dropout(p=0.09197568972148612, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Sigmoid()
    (8): Dropout(p=0.09197568972148612, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Sigmoid()
    (11): Dropout(p=0.09197568972148612, inplace=False)
```

```
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc             0.09540636092424393
         val_loss             2.3938796520233154
        valid_mapk            0.19680748879909515


train_model result: {'valid_mapk': 0.19680748879909515, 'val_loss': 2.3938796520233154, 'val_

spotPython tuning: 2.391202926635742 [###-------] 33.73%


config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': Sigmoid(), 'optimizer': 'Adamax'
model: NetLightBase(
  (act_fn): Sigmoid()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
```

```
    (1): Sigmoid()
    (2): Dropout(p=0.09402840843688196, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Sigmoid()
    (5): Dropout(p=0.09402840843688196, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Sigmoid()
    (8): Dropout(p=0.09402840843688196, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Sigmoid()
    (11): Dropout(p=0.09402840843688196, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc             0.07773851603269577
         val_loss             2.3984711170196533
        valid_mapk            0.1778227835893631


train_model result: {'valid_mapk': 0.1778227835893631, 'val_loss': 2.3984711170196533, 'val_a

spotPython tuning: 2.391202926635742 [####------] 35.43%
```

```
config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': Sigmoid(), 'optimizer': 'Adamax'
model: NetLightBase(
  (act_fn): Sigmoid()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Sigmoid()
    (2): Dropout(p=0.09268516166956951, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Sigmoid()
    (5): Dropout(p=0.09268516166956951, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Sigmoid()
    (8): Dropout(p=0.09268516166956951, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Sigmoid()
    (11): Dropout(p=0.09268516166956951, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

```
       Validate metric             DataLoader 0

           val_acc             0.07773851603269577
          val_loss             2.4029440879821777
         valid_mapk            0.17020319402217865


train_model result: {'valid_mapk': 0.17020319402217865, 'val_loss': 2.4029440879821777, 'val_

spotPython tuning: 2.391202926635742 [####------] 37.12%


config: {'l1': 4, 'epochs': 2, 'batch_size': 128, 'act_fn': Tanh(), 'optimizer': 'AdamW', 'd
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=4, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.0014418145120956294, inplace=False)
    (3): Linear(in_features=4, out_features=2, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.0014418145120956294, inplace=False)
    (6): Linear(in_features=2, out_features=2, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.0014418145120956294, inplace=False)
    (9): Linear(in_features=2, out_features=1, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.0014418145120956294, inplace=False)
    (12): Linear(in_features=1, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric              DataLoader 0

          val_acc             0.08127208799123764
         val_loss              2.400625705718994
        valid_mapk            0.12408372014760971


train_model result: {'valid_mapk': 0.12408372014760971, 'val_loss': 2.400625705718994, 'val_a

spotPython tuning: 2.391202926635742 [####------] 38.66%


config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': Tanh(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.07477269923002072, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.07477269923002072, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.07477269923002072, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.07477269923002072, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]
```

```
Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric              DataLoader 0

          val_acc              0.1342756152153015
         val_loss               2.395336627960205
        valid_mapk             0.2142811268568039


train_model result: {'valid_mapk': 0.2142811268568039, 'val_loss': 2.395336627960205, 'val_ac

spotPython tuning: 2.391202926635742 [####------] 40.34%


config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': Tanh(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.07337911615827276, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.07337911615827276, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
```

```
    (8): Dropout(p=0.07337911615827276, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.07337911615827276, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc             0.08127208799123764
         val_loss             2.3992538452148438
        valid_mapk            0.1687082052230835


train_model result: {'valid_mapk': 0.1687082052230835, 'val_loss': 2.3992538452148438, 'val_a

spotPython tuning: 2.391202926635742 [####------] 42.13%
```

```
config: {'l1': 8, 'epochs': 2, 'batch_size': 256, 'act_fn': Tanh(), 'optimizer': 'AdamW', 'd
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.08268854046625816, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.08268854046625816, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.08268854046625816, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.08268854046625816, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

Validate metric	DataLoader 0
val_acc	0.09540636092424393
val_loss	2.398329734802246
valid_mapk	0.17837336659431458

```
train_model result: {'valid_mapk': 0.17837336659431458, 'val_loss': 2.398329734802246, 'val_a

spotPython tuning: 2.391202926635742 [####------] 43.70%


config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': Tanh(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.09469171302595729, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.09469171302595729, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.09469171302595729, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.09469171302595729, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]


      Validate metric              DataLoader 0

          val_acc              0.0989399328827858
         val_loss              2.3968310356140137
        valid_mapk             0.18629436194896698


train_model result: {'valid_mapk': 0.18629436194896698, 'val_loss': 2.3968310356140137, 'val_

spotPython tuning: 2.391202926635742 [#####-----] 46.06%


config: {'l1': 4, 'epochs': 4, 'batch_size': 128, 'act_fn': Sigmoid(), 'optimizer': 'Adamax'
model: NetLightBase(
  (act_fn): Sigmoid()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=4, bias=True)
    (1): Sigmoid()
    (2): Dropout(p=0.049693804506661804, inplace=False)
    (3): Linear(in_features=4, out_features=2, bias=True)
    (4): Sigmoid()
    (5): Dropout(p=0.049693804506661804, inplace=False)
    (6): Linear(in_features=2, out_features=2, bias=True)
    (7): Sigmoid()
    (8): Dropout(p=0.049693804506661804, inplace=False)
    (9): Linear(in_features=2, out_features=1, bias=True)
    (10): Sigmoid()
    (11): Dropout(p=0.049693804506661804, inplace=False)
    (12): Linear(in_features=1, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric              DataLoader 0

          val_acc              0.07773851603269577
         val_loss              2.3979310989379883
        valid_mapk             0.1463155895471573


train_model result: {'valid_mapk': 0.1463155895471573, 'val_loss': 2.3979310989379883, 'val_a

spotPython tuning: 2.391202926635742 [#####-----] 48.53%


config: {'l1': 8, 'epochs': 4, 'batch_size': 128, 'act_fn': Tanh(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.0838150312090968, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.0838150312090968, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.0838150312090968, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
```

```
    (10): Tanh()
    (11): Dropout(p=0.0838150312090968, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc              0.07773851603269577
         val_loss               2.401747465133667
        valid_mapk             0.15083268284797668


train_model result: {'valid_mapk': 0.15083268284797668, 'val_loss': 2.401747465133667, 'val_a

spotPython tuning: 2.391202926635742 [#####-----] 50.56%


config: {'l1': 8, 'epochs': 2, 'batch_size': 64, 'act_fn': Sigmoid(), 'optimizer': 'Adam', 'd
model: NetLightBase(
  (act_fn): Sigmoid()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
```

```
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Sigmoid()
    (2): Dropout(p=0.08817153110926251, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Sigmoid()
    (5): Dropout(p=0.08817153110926251, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Sigmoid()
    (8): Dropout(p=0.08817153110926251, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Sigmoid()
    (11): Dropout(p=0.08817153110926251, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc            0.12720848619937897
         val_loss             2.393893003463745
        valid_mapk            0.189236119389534


train_model result: {'valid_mapk': 0.189236119389534, 'val_loss': 2.393893003463745, 'val_acc

spotPython tuning: 2.391202926635742 [#####-----] 52.45%
```

```
config: {'l1': 8, 'epochs': 2, 'batch_size': 64, 'act_fn': ReLU(), 'optimizer': 'Adam', 'drop
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.07771566300016146, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.07771566300016146, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.07771566300016146, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.07771566300016146, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

Validate metric	DataLoader 0
val_acc	0.11307420581579208
val_loss	2.3967034816741943
valid_mapk	0.19697144627571106

```
train_model result: {'valid_mapk': 0.19697144627571106, 'val_loss': 2.3967034816741943, 'val_

spotPython tuning: 2.391202926635742 [#####-----] 54.27%


config: {'l1': 8, 'epochs': 2, 'batch_size': 128, 'act_fn': ReLU(), 'optimizer': 'AdamW', 'd
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.06226910669369606, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.06226910669369606, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.06226910669369606, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.06226910669369606, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

```
       Validate metric             DataLoader 0

           val_acc              0.0742049440741539
          val_loss              2.3978302478790283
         valid_mapk              0.167711541056633


train_model result: {'valid_mapk': 0.167711541056633, 'val_loss': 2.3978302478790283, 'val_ac

spotPython tuning: 2.391202926635742 [######----] 56.10%


config: {'l1': 8, 'epochs': 2, 'batch_size': 64, 'act_fn': ReLU(), 'optimizer': 'Adam', 'drop
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.06999018745556085, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.06999018745556085, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.06999018745556085, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.06999018745556085, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc             0.0989399328827858
         val_loss              2.397876262664795
        valid_mapk            0.1834876537322998


train_model result: {'valid_mapk': 0.1834876537322998, 'val_loss': 2.397876262664795, 'val_ac

spotPython tuning: 2.391202926635742 [######----] 58.16%


config: {'l1': 8, 'epochs': 2, 'batch_size': 64, 'act_fn': Tanh(), 'optimizer': 'Adam', 'drop
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.0833202759846963, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.0833202759846963, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.0833202759846963, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.0833202759846963, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]
```

```
Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc             0.13074204325675964
         val_loss              2.395012140274048
        valid_mapk            0.17737267911434174


train_model result: {'valid_mapk': 0.17737267911434174, 'val_loss': 2.395012140274048, 'val_a

spotPython tuning: 2.391202926635742 [######----] 60.21%


config: {'l1': 4, 'epochs': 2, 'batch_size': 64, 'act_fn': Sigmoid(), 'optimizer': 'Adamax',
model: NetLightBase(
  (act_fn): Sigmoid()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=4, bias=True)
    (1): Sigmoid()
    (2): Dropout(p=0.002663627086246678, inplace=False)
    (3): Linear(in_features=4, out_features=2, bias=True)
    (4): Sigmoid()
    (5): Dropout(p=0.002663627086246678, inplace=False)
    (6): Linear(in_features=2, out_features=2, bias=True)
    (7): Sigmoid()
    (8): Dropout(p=0.002663627086246678, inplace=False)
    (9): Linear(in_features=2, out_features=1, bias=True)
    (10): Sigmoid()
    (11): Dropout(p=0.002663627086246678, inplace=False)
```

```
    (12): Linear(in_features=1, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc              0.10954063385725021
         val_loss              2.3961658477783203
        valid_mapk             0.19604553282260895


train_model result: {'valid_mapk': 0.19604553282260895, 'val_loss': 2.3961658477783203, 'val_

spotPython tuning: 2.391202926635742 [######----] 62.21%


config: {'l1': 8, 'epochs': 2, 'batch_size': 128, 'act_fn': Tanh(), 'optimizer': 'AdamW', 'd
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.04509229207652661, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.04509229207652661, inplace=False)
```

```
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.04509229207652661, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.04509229207652661, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc              0.09540636092424393
         val_loss               2.395824909210205
        valid_mapk             0.18790186941623688


train_model result: {'valid_mapk': 0.18790186941623688, 'val_loss': 2.395824909210205, 'val_a

spotPython tuning: 2.391202926635742 [######----] 64.14%


config: {'l1': 8, 'epochs': 2, 'batch_size': 128, 'act_fn': Tanh(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
```

```
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.06302007424917173, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.06302007424917173, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.06302007424917173, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.06302007424917173, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc             0.0989399328827858
         val_loss              2.397738456726074
        valid_mapk            0.16311407089233398


train_model result: {'valid_mapk': 0.16311407089233398, 'val_loss': 2.397738456726074, 'val_a

spotPython tuning: 2.391202926635742 [#######---] 66.29%
```

```
config: {'l1': 8, 'epochs': 2, 'batch_size': 128, 'act_fn': Tanh(), 'optimizer': 'Adamax', 'c
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.036622338852388206, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.036622338852388206, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.036622338852388206, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.036622338852388206, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

Validate metric	DataLoader 0
val_acc	0.07067137956619263
val_loss	2.399679183959961
valid_mapk	0.1584201455116272

```
train_model result: {'valid_mapk': 0.1584201455116272, 'val_loss': 2.399679183959961, 'val_ac

spotPython tuning: 2.391202926635742 [#######---] 68.38%


config: {'l1': 8, 'epochs': 2, 'batch_size': 64, 'act_fn': Sigmoid(), 'optimizer': 'Adam', 'd
model: NetLightBase(
  (act_fn): Sigmoid()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Sigmoid()
    (2): Dropout(p=0.0828460725027117, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Sigmoid()
    (5): Dropout(p=0.0828460725027117, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Sigmoid()
    (8): Dropout(p=0.0828460725027117, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Sigmoid()
    (11): Dropout(p=0.0828460725027117, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

```
      Validate metric           DataLoader 0

          val_acc            0.09540636092424393
         val_loss            2.3998100757598877
        valid_mapk           0.1641203761100769


train_model result: {'valid_mapk': 0.1641203761100769, 'val_loss': 2.3998100757598877, 'val_a

spotPython tuning: 2.391202926635742 [#######---] 70.25%


config: {'l1': 8, 'epochs': 4, 'batch_size': 64, 'act_fn': ReLU(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): ReLU()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.08066494792071505, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.08066494792071505, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.08066494792071505, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.08066494792071505, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric              DataLoader 0

          val_acc              0.10954063385725021
         val_loss              2.3938405513763428
        valid_mapk             0.21101465821266174


train_model result: {'valid_mapk': 0.21101465821266174, 'val_loss': 2.3938405513763428, 'val_

spotPython tuning: 2.391202926635742 [#######---] 73.01%


config: {'l1': 4, 'epochs': 4, 'batch_size': 64, 'act_fn': Sigmoid(), 'optimizer': 'AdamW',
model: NetLightBase(
  (act_fn): Sigmoid()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=4, bias=True)
    (1): Sigmoid()
    (2): Dropout(p=0.06401826146652642, inplace=False)
    (3): Linear(in_features=4, out_features=2, bias=True)
    (4): Sigmoid()
    (5): Dropout(p=0.06401826146652642, inplace=False)
    (6): Linear(in_features=2, out_features=2, bias=True)
    (7): Sigmoid()
    (8): Dropout(p=0.06401826146652642, inplace=False)
    (9): Linear(in_features=2, out_features=1, bias=True)
    (10): Sigmoid()
    (11): Dropout(p=0.06401826146652642, inplace=False)
```

```
    (12): Linear(in_features=1, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc             0.09187278896570206
         val_loss              2.399857997894287
        valid_mapk            0.1699845790863037


train_model result: {'valid_mapk': 0.1699845790863037, 'val_loss': 2.399857997894287, 'val_ac

spotPython tuning: 2.391202926635742 [########--] 75.57%


config: {'l1': 8, 'epochs': 4, 'batch_size': 64, 'act_fn': Tanh(), 'optimizer': 'Adamax', 'di
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
```

```
    (1): Tanh()
    (2): Dropout(p=0.0990383513029943, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.0990383513029943, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.0990383513029943, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.0990383513029943, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric            DataLoader 0

          val_acc             0.10600706934928894
         val_loss              2.390947103500366
        valid_mapk            0.21903935074806213


train_model result: {'valid_mapk': 0.21903935074806213, 'val_loss': 2.390947103500366, 'val_a

spotPython tuning: 2.390947103500366 [########--] 78.43%
```

```
config: {'l1': 4, 'epochs': 4, 'batch_size': 256, 'act_fn': Sigmoid(), 'optimizer': 'Adamax'
model: NetLightBase(
  (act_fn): Sigmoid()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=4, bias=True)
    (1): Sigmoid()
    (2): Dropout(p=0.04054496556325391, inplace=False)
    (3): Linear(in_features=4, out_features=2, bias=True)
    (4): Sigmoid()
    (5): Dropout(p=0.04054496556325391, inplace=False)
    (6): Linear(in_features=2, out_features=2, bias=True)
    (7): Sigmoid()
    (8): Dropout(p=0.04054496556325391, inplace=False)
    (9): Linear(in_features=2, out_features=1, bias=True)
    (10): Sigmoid()
    (11): Dropout(p=0.04054496556325391, inplace=False)
    (12): Linear(in_features=1, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

```
       Validate metric             DataLoader 0

           val_acc              0.14134275913238525
          val_loss              2.3954479694366455
         valid_mapk             0.19562596082687378
```

```
train_model result: {'valid_mapk': 0.19562596082687378, 'val_loss': 2.3954479694366455, 'val_
```

```
spotPython tuning: 2.390947103500366 [########--] 80.57%
```

```
config: {'l1': 8, 'epochs': 4, 'batch_size': 64, 'act_fn': Tanh(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.09100452995133981, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.09100452995133981, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.09100452995133981, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.09100452995133981, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)
```

```
Sanity Checking: 0it [00:00, ?it/s]
```

```
Training: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric            DataLoader 0

          val_acc             0.16254417598247528
         val_loss             2.3714849948883057
        valid_mapk            0.23244598507881165


train_model result: {'valid_mapk': 0.23244598507881165, 'val_loss': 2.3714849948883057, 'val_

spotPython tuning: 2.3714849948883057 [########--] 83.24%


config: {'l1': 4, 'epochs': 4, 'batch_size': 128, 'act_fn': Tanh(), 'optimizer': 'Adamax', '


model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=4, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.07122356722940353, inplace=False)
    (3): Linear(in_features=4, out_features=2, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.07122356722940353, inplace=False)
    (6): Linear(in_features=2, out_features=2, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.07122356722940353, inplace=False)
    (9): Linear(in_features=2, out_features=1, bias=True)
```

```
    (10): Tanh()
    (11): Dropout(p=0.07122356722940353, inplace=False)
    (12): Linear(in_features=1, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc              0.10954063385725021
         val_loss               2.396536111831665
        valid_mapk             0.21604937314987183


train_model result: {'valid_mapk': 0.21604937314987183, 'val_loss': 2.396536111831665, 'val_a

spotPython tuning: 2.3714849948883057 [#########-] 85.63%


config: {'l1': 8, 'epochs': 4, 'batch_size': 64, 'act_fn': Tanh(), 'optimizer': 'Adamax', 'di
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
```

```
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.09190899432519349, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.09190899432519349, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.09190899432519349, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.09190899432519349, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc             0.10600706934928894
         val_loss             2.3926639556884766
        valid_mapk            0.19637344777584076


train_model result: {'valid_mapk': 0.19637344777584076, 'val_loss': 2.3926639556884766, 'val_
```

```
spotPython tuning: 2.3714849948883057 [#########-] 88.02%


config: {'l1': 8, 'epochs': 2, 'batch_size': 64, 'act_fn': Tanh(), 'optimizer': 'Adamax', 'di
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.1, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.1, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.1, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.1, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

Validate metric	DataLoader 0
val_acc	0.11660777032375336

```
         val_loss              2.3946642875671387
        valid_mapk             0.19293980300426483


train_model result: {'valid_mapk': 0.19293980300426483, 'val_loss': 2.3946642875671387, 'val_

spotPython tuning: 2.3714849948883057 [#########-] 90.32%


config: {'l1': 8, 'epochs': 4, 'batch_size': 64, 'act_fn': Tanh(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.07666934610816932, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.07666934610816932, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.07666934610816932, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.07666934610816932, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]


Training: 0it [00:00, ?it/s]


Validation: 0it [00:00, ?it/s]


Validation: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc             0.15194346010684967
         val_loss             2.3836989402770996
        valid_mapk            0.2635031044483185


train_model result: {'valid_mapk': 0.2635031044483185, 'val_loss': 2.3836989402770996, 'val_a

spotPython tuning: 2.3714849948883057 [#########-] 92.68%


config: {'l1': 8, 'epochs': 4, 'batch_size': 64, 'act_fn': Tanh(), 'optimizer': 'Adamax', 'di
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.08289084581096985, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.08289084581096985, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.08289084581096985, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.08289084581096985, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)
```

```
Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric              DataLoader 0

          val_acc              0.10954063385725021
         val_loss               2.386335611343384
        valid_mapk             0.21199843287467957


train_model result: {'valid_mapk': 0.21199843287467957, 'val_loss': 2.386335611343384, 'val_a

spotPython tuning: 2.3714849948883057 [##########] 95.10%


config: {'l1': 8, 'epochs': 2, 'batch_size': 128, 'act_fn': Tanh(), 'optimizer': 'Adamax', 'd
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.0818162756981457, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
```

```
    (5): Dropout(p=0.0818162756981457, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.0818162756981457, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.0818162756981457, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric              DataLoader 0

          val_acc              0.10600706934928894
         val_loss               2.394561767578125
        valid_mapk             0.21098573505878448


train_model result: {'valid_mapk': 0.21098573505878448, 'val_loss': 2.394561767578125, 'val_a

spotPython tuning: 2.3714849948883057 [##########] 96.93%


config: {'l1': 8, 'epochs': 4, 'batch_size': 64, 'act_fn': Tanh(), 'optimizer': 'Adamax', 'di
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
```

```
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.06946939697491468, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.06946939697491468, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.06946939697491468, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.06946939697491468, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc             0.11307420581579208
         val_loss              2.391633987426758
        valid_mapk            0.19023919105529785


train_model result: {'valid_mapk': 0.19023919105529785, 'val_loss': 2.391633987426758, 'val_a
```

```
spotPython tuning: 2.3714849948883057 [##########] 100.00% Done...
```

```
<spotPython.spot.spot.Spot at 0x2b8f0bf10>
```

21.9 Step 9: Tensorboard

The textual output shown in the console (or code cell) can be visualized with Tensorboard as described in Section 14.9, see also the description in the documentation: Tensorboard.

21.10 Step 10: Results

After the hyperparameter tuning run is finished, the results can be analyzed as described in Section 14.10.

```
spot_tuner.plot_progress(log_y=False,
    filename="./figures/" + experiment_name+"_progress.png")
```

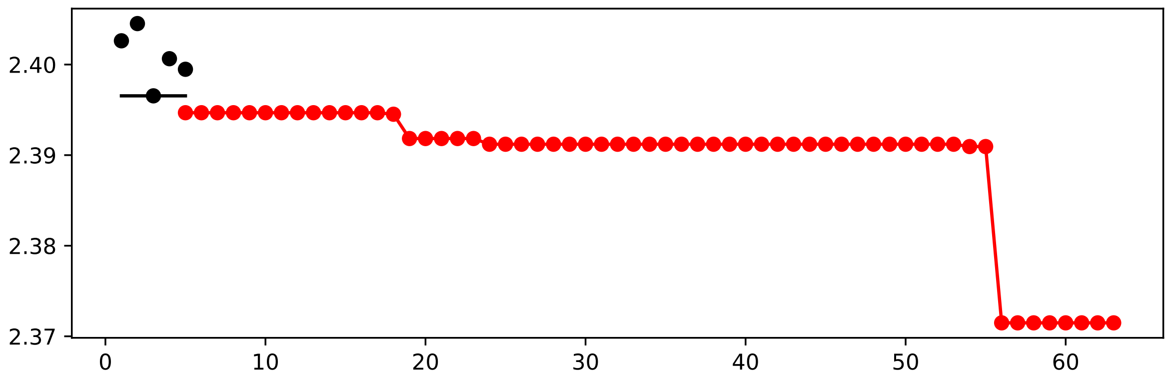


Figure 21.1: Progress plot. *Black* dots denote results from the initial design. *Red* dots illustrate the improvement found by the surrogate model based optimization.

```
from spotPython.utils.eda import gen_design_table
print(gen_design_table(fun_control=fun_control, spot=spot_tuner))
```

```
| name           | type    | default    |    lower |    upper |                 tuned | transform
|----------------|---------|------------|----------|----------|-----------------------|-------------
| l1             | int     | 3          |      2.0 |      3.0 |                   3.0 | transform_pow
```

```
| epochs       | int    | 4         |     1.0 |    2.0 |                 2.0 | transform_pow
| batch_size   | int    | 4         |     6.0 |    8.0 |                 6.0 | transform_pow
| act_fn       | factor | ReLU      |     0.0 |    2.0 |                 1.0 | None
| optimizer    | factor | SGD       |     0.0 |    3.0 |                 2.0 | None
| dropout_prob | float  | 0.01      |     0.0 |    0.1 | 0.09100452995133981 | None
| lr_mult      | float  | 1.0       |     0.1 |   10.0 |    9.51342550806287 | None
```

```python
spot_tuner.plot_importance(threshold=0.025,
    filename="./figures/" + experiment_name+"_importance.png")
```

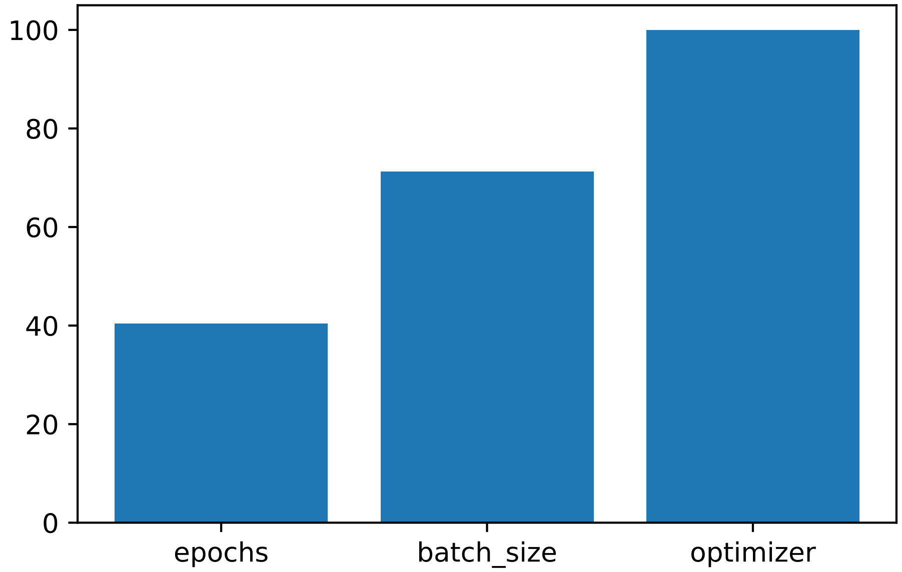


Figure 21.2: Variable importance plot, threshold 0.025.

21.10.1 Get the Tuned Architecture

```python
from spotPython.hyperparameters.values import get_one_config_from_X
X = spot_tuner.to_all_dim(spot_tuner.min_X.reshape(1,-1))
config = get_one_config_from_X(X, fun_control)
config
```

```
{'l1': 8,
 'epochs': 4,
 'batch_size': 64,
```

```
 'act_fn': Tanh(),
 'optimizer': 'Adamax',
 'dropout_prob': 0.09100452995133981,
 'lr_mult': 9.51342550806287}
```

- Test on the full data set

```
from spotPython.light.traintest import test_model
test_model(config, fun_control)
```

```
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.09100452995133981, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.09100452995133981, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.09100452995133981, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.09100452995133981, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)
```

```
Sanity Checking: 0it [00:00, ?it/s]
```

```
Training: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]
```

```
Testing: 0it [00:00, ?it/s]
```

```
        Test metric             DataLoader 0

      test_mapk_epoch          0.2736545205116272
          val_acc              0.17538896203041077
         val_loss               2.385132074356079
```

```
test_model result: {'test_mapk_epoch': 0.2736545205116272, 'val_loss': 2.385132074356079, 'va
```

```
(2.385132074356079, 0.17538896203041077)
```

21.10.2 Cross Validation With Lightning

- The `KFold` class from `sklearn.model_selection` is used to generate the folds for cross-validation.
- These mechanism is used to generate the folds for the final evaluation of the model.
- The `CrossValidationDataModule` class [SOURCE] is used to generate the folds for the hyperparameter tuning process.
- It is called from the `cv_model` function [SOURCE].

```
from spotPython.light.traintest import cv_model
cv_model(config, fun_control)
```

```
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.09100452995133981, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.09100452995133981, inplace=False)
```

```
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.09100452995133981, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.09100452995133981, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)
k: 0
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.09100452995133981, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.09100452995133981, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.09100452995133981, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.09100452995133981, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric              DataLoader 0

          val_acc             0.056338027119636536
         val_loss               2.395859956741333
        valid_mapk             0.1229538768529892


train_model result: {'valid_mapk': 0.1229538768529892, 'val_loss': 2.395859956741333, 'val_ac
k: 1
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.09100452995133981, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.09100452995133981, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.09100452995133981, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.09100452995133981, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric            DataLoader 0

          val_acc             0.2535211145877838
         val_loss             2.3548269271850586
        valid_mapk            0.4419642984867096


train_model result: {'valid_mapk': 0.4419642984867096, 'val_loss': 2.3548269271850586, 'val_a
k: 2
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.09100452995133981, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.09100452995133981, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.09100452995133981, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.09100452995133981, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]
```

```
Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc              0.11267605423927307
         val_loss              2.3698813915252686
        valid_mapk             0.1856398731470108


train_model result: {'valid_mapk': 0.1856398731470108, 'val_loss': 2.3698813915252686, 'val_a
k: 3
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.09100452995133981, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.09100452995133981, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.09100452995133981, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.09100452995133981, inplace=False)
```

```
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc             0.2112676054239273
         val_loss             2.3213462829589844
        valid_mapk            0.2922247052192688


train_model result: {'valid_mapk': 0.2922247052192688, 'val_loss': 2.3213462829589844, 'val_a
k: 4
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.09100452995133981, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.09100452995133981, inplace=False)
```

```
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.09100452995133981, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.09100452995133981, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc              0.2112676054239273
         val_loss              2.3328769207000732
        valid_mapk             0.3534226417541504


train_model result: {'valid_mapk': 0.3534226417541504, 'val_loss': 2.3328769207000732, 'val_a
k: 5
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
```

```
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.09100452995133981, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.09100452995133981, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.09100452995133981, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.09100452995133981, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric            DataLoader 0

          val_acc             0.23943662643432617
         val_loss             2.3028900623321533
        valid_mapk            0.4484747052192688
```

```
train_model result: {'valid_mapk': 0.4484747052192688, 'val_loss': 2.3028900623321533, 'val_a
k: 6
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.09100452995133981, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.09100452995133981, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.09100452995133981, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.09100452995133981, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

```
     Validate metric             DataLoader 0

         val_acc              0.23943662643432617
        val_loss               2.307607889175415
       valid_mapk             0.4047619104385376


train_model result: {'valid_mapk': 0.4047619104385376, 'val_loss': 2.307607889175415, 'val_ac
k: 7
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.09100452995133981, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.09100452995133981, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.09100452995133981, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.09100452995133981, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric              DataLoader 0

          val_acc              0.2142857164144516
         val_loss               2.309537649154663
        valid_mapk             0.3689236044883728


train_model result: {'valid_mapk': 0.3689236044883728, 'val_loss': 2.309537649154663, 'val_ac
k: 8
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.09100452995133981, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.09100452995133981, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.09100452995133981, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.09100452995133981, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric             DataLoader 0

          val_acc             0.24285714328289032
         val_loss              2.290733814239502
        valid_mapk            0.4131944179534912


train_model result: {'valid_mapk': 0.4131944179534912, 'val_loss': 2.290733814239502, 'val_ac
k: 9
model: NetLightBase(
  (act_fn): Tanh()
  (train_mapk): MAPK()
  (valid_mapk): MAPK()
  (test_mapk): MAPK()
  (model): Sequential(
    (0): Linear(in_features=64, out_features=8, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.09100452995133981, inplace=False)
    (3): Linear(in_features=8, out_features=4, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.09100452995133981, inplace=False)
    (6): Linear(in_features=4, out_features=4, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.09100452995133981, inplace=False)
    (9): Linear(in_features=4, out_features=2, bias=True)
    (10): Tanh()
    (11): Dropout(p=0.09100452995133981, inplace=False)
    (12): Linear(in_features=2, out_features=11, bias=True)
  )
)

Sanity Checking: 0it [00:00, ?it/s]
```

```
Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]


      Validate metric           DataLoader 0

          val_acc             0.2571428716182709
         val_loss              2.28702712059021
        valid_mapk            0.3311631977558136


train_model result: {'valid_mapk': 0.3311631977558136, 'val_loss': 2.28702712059021, 'val_ac
cv_model mapk result: 0.3362723231315613

0.3362723231315613
```

> **Note: Evaluation for the Final Comaprison**
>
> - This is the evaluation that will be used in the comparison.

21.10.3 Detailed Hyperparameter Plots

```python
filename = "./figures/" + experiment_name
spot_tuner.plot_important_hyperparameter_contour(filename=filename)
```

```
epochs:  40.37531278681481
batch_size:  71.29681001107211
optimizer:  100.0
```

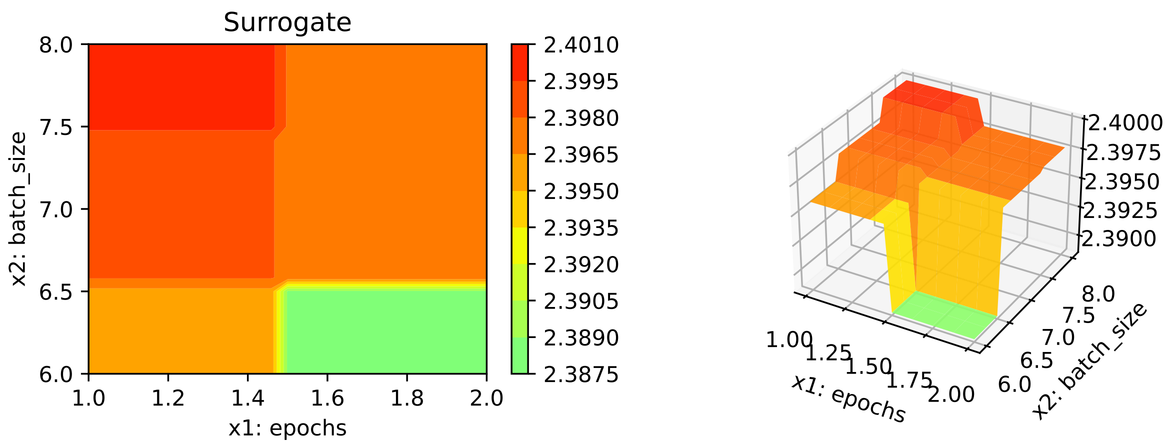
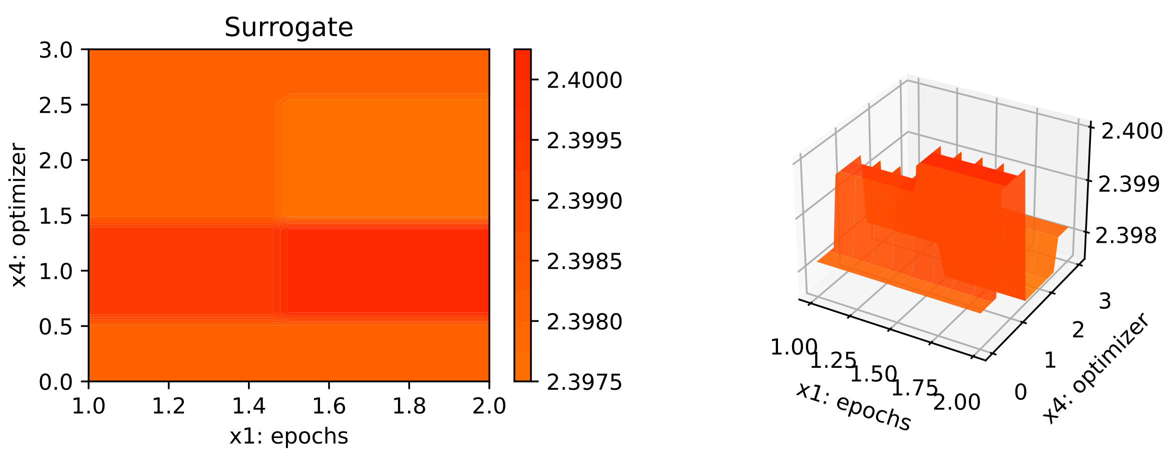
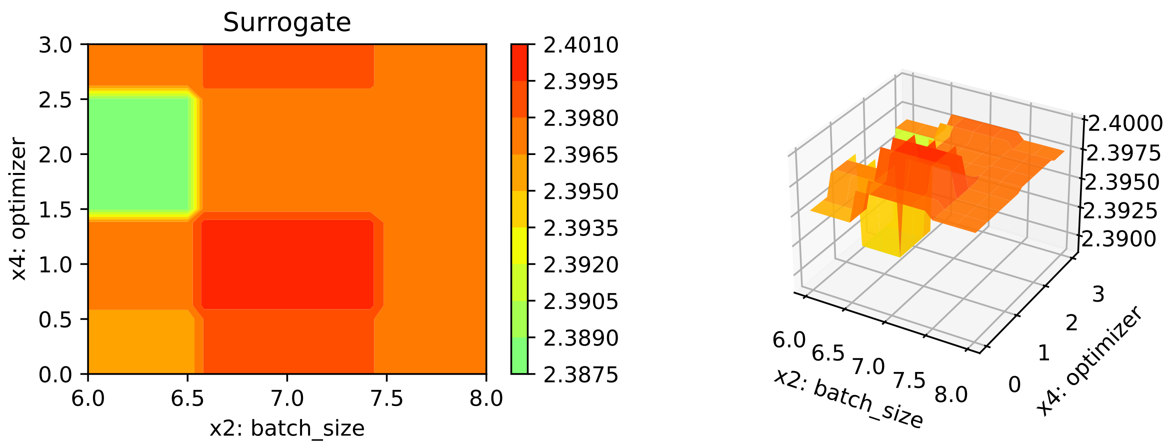


Figure 21.3: Contour plots.





21.10.4 Parallel Coordinates Plot

```
spot_tuner.parallel_plot()
```

```
Unable to display output for mime type(s): text/html
```

Parallel coordinates plots

```
Unable to display output for mime type(s): text/html
```

21.10.5 Plot all Combinations of Hyperparameters

- Warning: this may take a while.

```
PLOT_ALL = False
if PLOT_ALL:
    n = spot_tuner.k
    for i in range(n-1):
        for j in range(i+1, n):
            spot_tuner.plot_contour(i=i, j=j, min_z=min_z, max_z = max_z)
```

22 Documentation of the Sequential Parameter Optimization

This document describes the `Spot` features.

22.1 Example: spot

```
import numpy as np
from math import inf
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
from scipy.optimize import shgo
from scipy.optimize import direct
from scipy.optimize import differential_evolution
import matplotlib.pyplot as plt
```

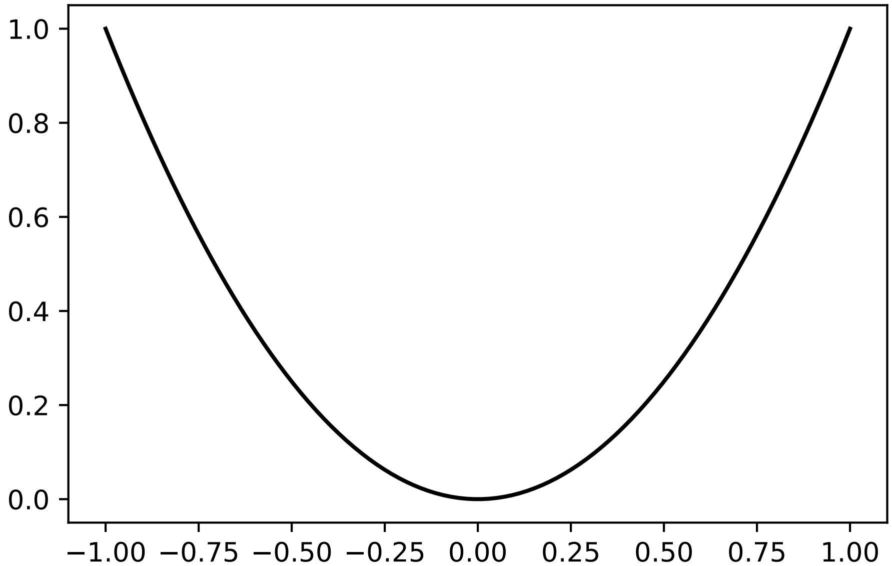
22.1.1 The Objective Function

The `spotPython` package provides several classes of objective functions. We will use an analytical objective function, i.e., a function that can be described by a (closed) formula:

$$f(x) = x^2$$

```
fun = analytical().fun_sphere
```

```
x = np.linspace(-1,1,100).reshape(-1,1)
y = fun(x)
plt.figure()
plt.plot(x,y, "k")
plt.show()
```




```
spot_1 = spot.Spot(fun=fun,
                   lower = np.array([-10]),
                   upper = np.array([100]),
                   fun_evals = 7,
                   fun_repeats = 1,
                   max_time = inf,
                   noise = False,
                   tolerance_x = np.sqrt(np.spacing(1)),
                   var_type=["num"],
                   infill_criterion = "y",
                   n_points = 1,
                   seed=123,
                   log_level = 50,
                   show_models=True,
                   fun_control = {},
                   design_control={"init_size": 5,
                                   "repeats": 1},
                   surrogate_control={"noise": False,
                                      "cod_type": "norm",
                                      "min_theta": -4,
                                      "max_theta": 3,
                                      "n_theta": 1,
                                      "model_optimizer": differential_evolution,
                                      "model_fun_evals": 1000,
```

```
})
```

`spot`'s `__init__` method sets the control parameters. There are two parameter groups:

1. external parameters can be specified by the user
2. internal parameters, which are handled by `spot`.

22.1.2 External Parameters

external parameter	type	description	default	mandatory
`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

22.2 The fun_control Dictionary

external parameter	type	description	default	mandatory
sigma	float	noise: standard deviation	0	yes
seed	int	seed for rng	124	yes

22.3 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

22.4 The `surrogate_control` Dictionary

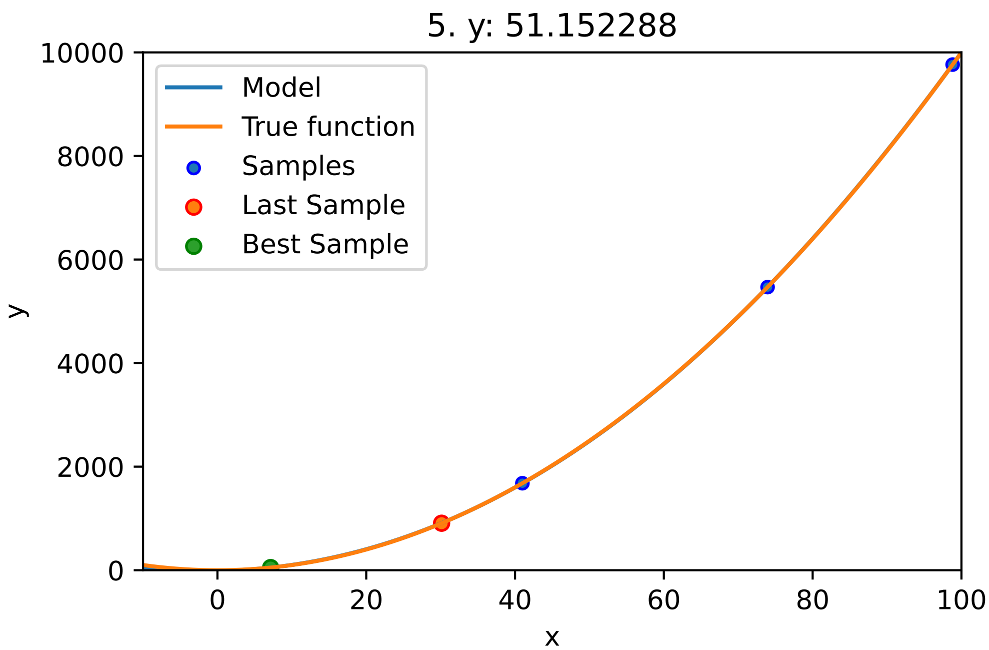
external parameter	type	description	default	mandatory
`noise`				
`model_optimizer`	object	optimizer	`differential_evolution`	no
`model_fun_evals`				
`min_theta`			`-3.`	
`max_theta`			`3.`	
`n_theta`			`1`	
`n_p`			`1`	
`optim_p`			`False`	
`cod_type`			`"norm"`	
`var_type`				
`use_cod_y`	bool		`False`	

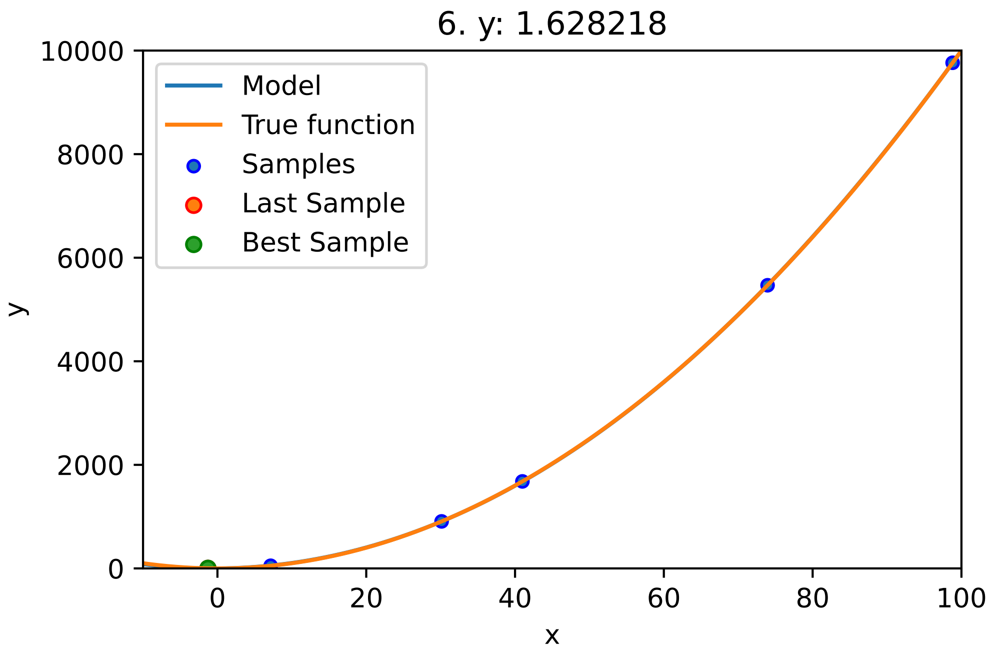
22.5 The `optimizer_control` Dictionary

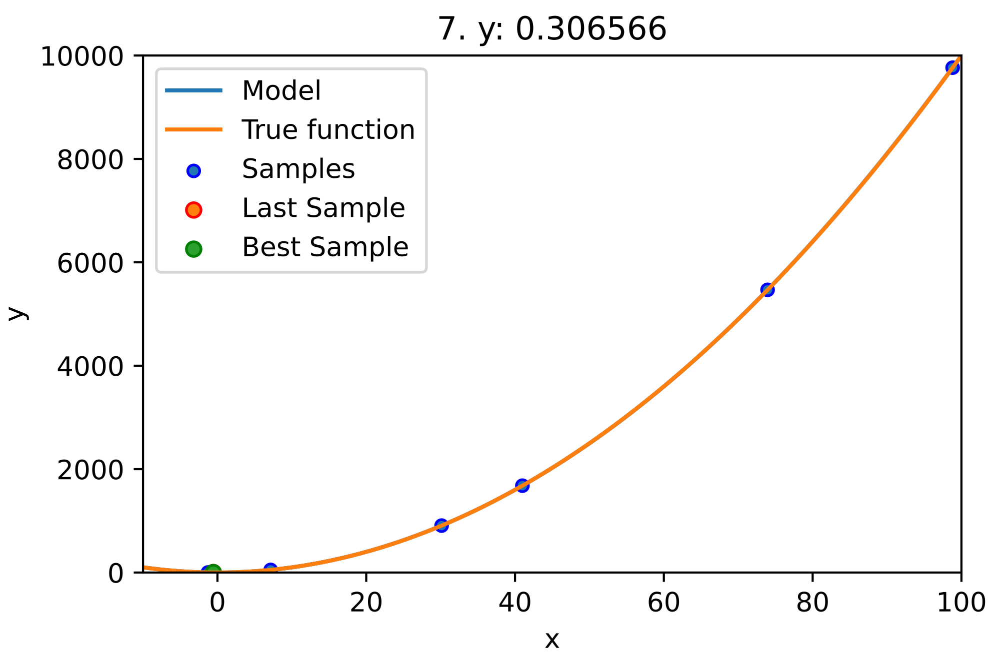
external parameter	type	description	default	mandatory
`max_iter`	int	max number of iterations. Note: these are the cheap evaluations on the surrogate.	`1000`	no

22.6 Run

```
spot_1.run()
```








```
<spotPython.spot.spot.Spot at 0x17f28aad0>
```

22.7 Print the Results

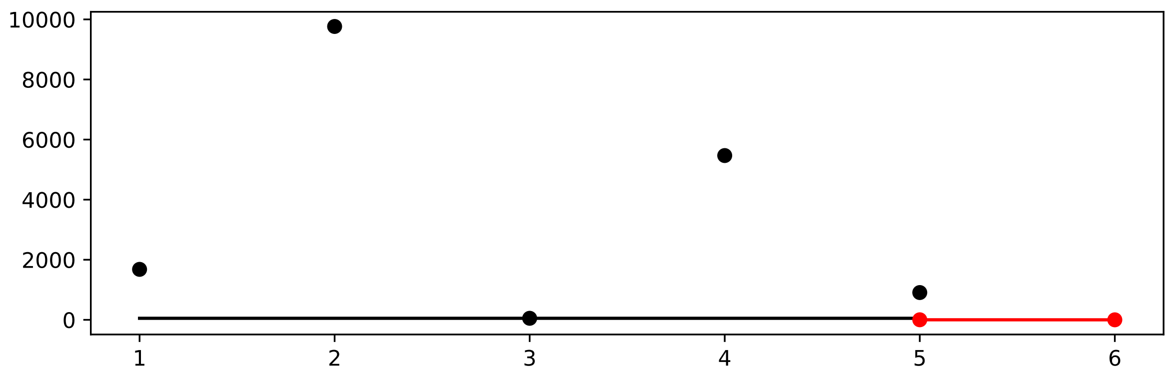
```
spot_1.print_results()
```

```
min y: 0.30656551286610595
x0: -0.5536835855126157
```

```
[['x0', -0.5536835855126157]]
```

22.8 Show the Progress

```
spot_1.plot_progress()
```

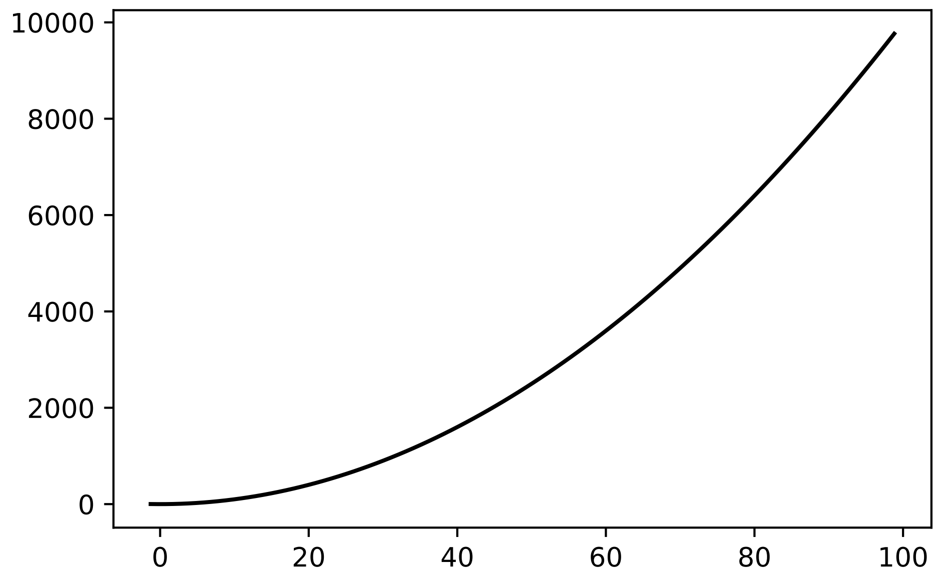


22.9 Visualize the Surrogate

- The plot method of the `kriging` surrogate is used.
- Note: the plot uses the interval defined by the ranges of the natural variables.

```
spot_1.surrogate.plot()
```

```
<Figure size 2700x1800 with 0 Axes>
```



22.10 Init: Build Initial Design

```
from spotPython.design.spacefilling import spacefilling
from spotPython.build.kriging import Kriging
from spotPython.fun.objectivefunctions import analytical
gen = spacefilling(2)
rng = np.random.RandomState(1)
lower = np.array([-5,-0])
upper = np.array([10,15])
fun = analytical().fun_branin
fun_control = {"sigma": 0,
               "seed": 123}

X = gen.scipy_lhd(10, lower=lower, upper = upper)
print(X)
y = fun(X, fun_control=fun_control)
print(y)
```

```
[[ 8.97647221 13.41926847]
 [ 0.66946019  1.22344228]
 [ 5.23614115 13.78185824]
 [ 5.6149825  11.5851384 ]
```

```
 [-1.72963184  1.66516096]
 [-4.26945568  7.1325531 ]
 [ 1.26363761 10.17935555]
 [ 2.88779942  8.05508969]
 [-3.39111089  4.15213772]
 [ 7.30131231  5.22275244]]
[128.95676449  31.73474356 172.89678121 126.71295908  64.34349975
  70.16178611  48.71407916  31.77322887  76.91788181  30.69410529]
```

22.11 Replicability

Seed

```
gen = spacefilling(2, seed=123)
X0 = gen.scipy_lhd(3)
gen = spacefilling(2, seed=345)
X1 = gen.scipy_lhd(3)
X2 = gen.scipy_lhd(3)
gen = spacefilling(2, seed=123)
X3 = gen.scipy_lhd(3)
X0, X1, X2, X3
```

```
(array([[0.77254938, 0.31539299],
        [0.59321338, 0.93854273],
        [0.27469803, 0.3959685 ]]),
 array([[0.78373509, 0.86811887],
        [0.06692621, 0.6058029 ],
        [0.41374778, 0.00525456]]),
 array([[0.121357  , 0.69043832],
        [0.41906219, 0.32838498],
        [0.86742658, 0.52910374]]),
 array([[0.77254938, 0.31539299],
        [0.59321338, 0.93854273],
        [0.27469803, 0.3959685 ]]))
```

22.12 Surrogates

22.12.1 A Simple Predictor

The code below shows how to use a simple model for prediction. Assume that only two (very costly) measurements are available:

1. f(0) = 0.5
2. f(2) = 2.5

We are interested in the value at $x_0 = 1$, i.e., $f(x_0 = 1)$, but cannot run an additional, third experiment.

```
from sklearn import linear_model
X = np.array([[0], [2]])
y = np.array([0.5, 2.5])
S_lm = linear_model.LinearRegression()
S_lm = S_lm.fit(X, y)
X0 = np.array([[1]])
y0 = S_lm.predict(X0)
print(y0)
```

```
[1.5]
```

Central Idea: Evaluation of the surrogate model `S_lm` is much cheaper (or / and much faster) than running the real-world experiment f.

22.13 Demo/Test: Objective Function Fails

SPOT expects `np.nan` values from failed objective function values. These are handled. Note: SPOT's counter considers only successful executions of the objective function.

```
import numpy as np
from spotPython.fun.objectivefunctions import analytical
from spotPython.spot import spot
import numpy as np
from math import inf
# number of initial points:
ni = 20
# number of points
n = 30
```

```
fun = analytical().fun_random_error
lower = np.array([-1])
upper = np.array([1])
design_control={"init_size": ni}

spot_1 = spot.Spot(fun=fun,
            lower = lower,
            upper= upper,
            fun_evals = n,
            show_progress=False,
            design_control=design_control,)
spot_1.run()
# To check whether the run was successfully completed,
# we compare the number of evaluated points to the specified
# number of points.
assert spot_1.y.shape[0] == n
```

```
[ 0.53176481 -0.9053821          nan -0.21843718  0.78240941 -0.58120945
 -0.3923345   0.67234256  0.31802454 -0.68898927 -0.75129705  0.97550354
  0.41757584  0.0786237          nan  0.23700598 -0.49274073 -0.82319082
 -0.17991251  0.1481835 ]
[-1.]

[0.95541987]

[0.17335968]

[-0.58552368]

[-0.20126111]

[-0.60100809]

[nan]

[-0.97897336]
```

```
[-0.2748985]


[0.8359486]

[0.99035591]
[0.01641232]

[0.5629346]
```

22.14 PyTorch: Detailed Description of the Data Splitting

22.14.1 Description of the "train_hold_out" Setting

The "train_hold_out" setting is used by default. It uses the loss function 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
```

22.14.1.1 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.

22.14.1.2 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`.

22.14.1.3 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.

22.14.1.4 Detailed Description of the Final Model Training and Evaluation

There are two methods that can be used for the final evaluation of a Pytorch model:

1. `"train_tuned` and
2. `"test_tuned"`.

`train_tuned()` is just a wrapper to `evaluate_hold_out` using the `train` data set. It is implemented as follows:

```
def train_tuned(
    net,
    train_dataset,
    shuffle,
    loss_function,
    metric,
```

```
        device=None,
        show_batch_interval=10_000,
        path=None,
        task=None,
        writer=None,
):
    evaluate_hold_out(
        net=net,
        train_dataset=train_dataset,
        shuffle=shuffle,
        test_dataset=None,
        loss_function=loss_function,
        metric=metric,
        device=device,
        show_batch_interval=show_batch_interval,
        path=path,
        task=task,
        writer=writer,
    )
```

The `test_tuned()` procedure is implemented as follows:

```
def test_tuned(net, shuffle, test_dataset=None, loss_function=None,
    metric=None, device=None, path=None, task=None):
    batch_size_instance = net.batch_size
    removed_attributes, net = get_removed_attributes_and_base_net(net)
    if path is not None:
        net.load_state_dict(torch.load(path))
        net.eval()
    try:
        device = getDevice(device=device)
        if torch.cuda.is_available():
            device = "cuda:0"
            if torch.cuda.device_count() > 1:
                print("We will use", torch.cuda.device_count(), "GPUs!")
                net = nn.DataParallel(net)
        net.to(device)
        valloader = torch.utils.data.DataLoader(
            test_dataset, batch_size=int(batch_size_instance),
            shuffle=shuffle,
            num_workers=0
        )
```

```
        metric_value, loss = validate_one_epoch(
            net, valloader=valloader, loss_function=loss_function,
            metric=metric, device=device, task=task
        )
        df_eval = loss
        df_metric = metric_value
        df_preds = np.nan
    except Exception as err:
        print(f"Error in Net_Core. Call to test_tuned() failed. {err=},
            {type(err)=}")
        df_eval = np.nan
        df_metric = np.nan
        df_preds = np.nan
    add_attributes(net, removed_attributes)
    print(f"Final evaluation: Validation loss: {df_eval}")
    print(f"Final evaluation: Validation metric: {df_metric}")
    print("----------------------------------------------")
    return df_eval, df_preds, df_metric
```

References

Bartz, Eva, Thomas Bartz-Beielstein, Martin Zaefferer, and Olaf Mersmann, eds. 2022. *Hyperparameter Tuning for Machine and Deep Learning with R - A Practical Guide*. Springer.

Bartz-Beielstein, Thomas. 2023. "PyTorch Hyperparameter Tuning with SPOT: Comparison with Ray Tuner and Default Hyperparameters on CIFAR10." https://github.com/sequential-parameter-optimization/spotPython/blob/main/notebooks/14_spot_ray_hpt_torch_cifar10.ipynb.

Bartz-Beielstein, Thomas, Jürgen Branke, Jörn Mehnen, and Olaf Mersmann. 2014. "Evolutionary Algorithms." *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 4 (3): 178–95.

Bartz-Beielstein, Thomas, Carola Doerr, Jakob Bossek, Sowmya Chandrasekaran, Tome Eftimov, Andreas Fischbach, Pascal Kerschke, et al. 2020. "Benchmarking in Optimization: Best Practice and Open Issues." arXiv. https://arxiv.org/abs/2007.03488.

Bartz-Beielstein, Thomas, Christian Lasarczyk, and Mike Preuss. 2005. "Sequential Parameter Optimization." In *Proceedings 2005 Congress on Evolutionary Computation (CEC'05), Edinburgh, Scotland*, edited by B McKay et al., 773–80. Piscataway NJ: IEEE Press.

Lewis, R M, V Torczon, and M W Trosset. 2000. "Direct search methods: Then and now." *Journal of Computational and Applied Mathematics* 124 (1–2): 191–207.

Li, Lisha, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2016. "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization." *arXiv e-Prints*, March, arXiv:1603.06560.

Meignan, David, Sigrid Knust, Jean-Marc Frayet, Gilles Pesant, and Nicolas Gaud. 2015. "A Review and Taxonomy of Interactive Optimization Methods in Operations Research." *ACM Transactions on Interactive Intelligent Systems*, September.

Montiel, Jacob, Max Halford, Saulo Martiello Mastelini, Geoffrey Bolmier, Raphael Sourty, Robin Vaysse, Adil Zouitine, et al. 2021. "River: Machine Learning for Streaming Data in Python."

PyTorch. 2023a. "Hyperparameter Tuning with Ray Tune." https://pytorch.org/tutorials/beginner/hyperparameter_tuning_tutorial.html.

———. 2023b. "Training a Classifier." https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html.