Hyperparameter Optimization and Successive Halving

AutoML – Assignment 1

Leiden University

The objective of this assignment is to provide a comprehensive understanding of surrogate models used for **H**yper**p**arameter **O**ptimization (HPO) and compare their performance with traditional methods like Grid Search and Random Search. You will be provided with a dataset of empirical learning curves collected with an updated version of **L**earning **C**urve **D**ata**b**ase (LCDB) [2]. Using this collected data you will be tasked with training a surrogate model that predicts the performance of a given configuration, and this will be used for running several hyperparameter optimisation techniques, i.e., Bayesian optimisation and successive halving. Code for random search will be provided.

1 Surrogate model

Hyperparameter optimisation can be an expensive operation, because of the individual models that need to be trained. To make the development and implementation of hyperparameter optimisation methods less expensive, often surrogate models are being employed. These are models that are trained on prior performance data of configurations.

To simulate the HPO in a resource-constraint environment, you will utilize a surrogate model trained on LCDB learning curve data to predict the performance of the sampled configurations rather than training a model to get the true performance.

Please note that for your set of experiments, pairs of machine learning algorithms and datasets require independent surrogate models. Since the machine learning algorithm to be optimized is always *KNN*, for each experimented dataset you need a uniquely/independently trained surrogate model. For each configuration, we have recorded the performance across various anchors (training set sizes). Normal HPO methods, such as random search and Bayesian optimisation, only use the full training set size (i.e., the final anchor). Methods such as successive halving also need performances of lower anchor sizes.

The best known package for defining a configuration space is the `ConfigSpace` package (available on PyPI), which allows to define a configuration space. We have also provided a configuration space (which you don't need for building the surrogate model). Familiarise yourself with the content of this package, as it will be important during the course.[1]

We have provided you with three important files:

1. `surrogate_model.py`: the main class handling the surrogate model. Two functions need to be implemented to make it work. To get a feeling of how this would work, you can already have a look at `example_run_experiment.py` (which interacts with this file).

2. `random_search.py`: in this file, we have provided code for the random search algorithm. Nop adjustments are needed, and it should work out of the box.

[1] Documentation: `https://automl.github.io/ConfigSpace/latest/`

3. `example_run_experiment.py`: an example how to interact with the API of the surrogate model and random search. Once the surrogate model has been implemented, this should work out of the box as well.

Complete the functions in `surrogate_model.py`, and ensure that you understand the workings of the framework.

2 Hyperparameter Optimization

2.1 SMBO implementation

In this assignment, we will see how to program an instantiation of Bayesian Optimization, in particular Sequential Model-based Optimization (SMBO). SMBO is a simple yet powerful technique, which is described in [1, 3]. SMBO can be written as follows, using various auxiliary functions in the data structure.

```
smbo = SequentialModelBasedOptimization()
smbo.initialize(list(...))

while budget left:
    smbo.fit_model()
    theta_new = smbo.select_configuration(sample_configurations(many))
    performance = optimizee(theta_new)
    smbo.update_runs((theta_new, performance))
```

IMPORTANT: For all hyperparameter optimization methods (SMBO, Random Search, and Grid Search) you should use the external surrogate model, trained on LCDB data, to predict the candidate configurations performances and use that as ground truth.

To aid you in the process we are providing a python script (`smbo.py`). Please investigate this file and proceed to complete the auxiliary functions, necessary for HPO execution using SMBO. These functions are listed hereafter:

- `select_configuration(...)`: Receives an array of configurations, and needs to select the best of these (i.e., the one that most likely yields the most improvement over the current best configuration, θ_{inc}). It uses an auxiliary function to calculate the expected improvement for each configuration.

- `expected_improvement(...)`: Function to determine for a set of configurations what the expected improvement is. Hint: the `GaussianProcessRegressor` has a predict function that takes as argument `return_std`, to receive both the predicted mean (μ) and standard deviation (σ) per configuration. EI is defined as:

$$EI(\mu, \sigma) = (f * -\mu)\Phi(\frac{f * -\mu}{\sigma}) + \sigma(\phi(\frac{f * -\mu}{\sigma}))$$

  where Φ and ϕ are the CDF and PDF of a standard normal distribution, respectively, $f*$ is the best seen configuration so far, and μ, σ are the predicted mean and standard deviation of the configuration for which expected improvement is being calculated. (Note a slightly different formulation, as we are optimizing).

- `update_runs(...)`: Replacement of intensification function. Intensify can only be used when working across multiple random seeds, cross-validation folds, and in this case, we have only a single measurement. Intensification will therefore yield no improvement over just running each configuration. This function adds the last run to the list of all seen runs (so the model can be trained on it) and updates θ_{inc} in case a new best algorithm is found.

Note that for each of the above functions that need to be implemented, you need to write approximately 3-10 lines of code.

Please ensure to run the code as much as possible according to the framework that is lined out in `example_run_experiment.py`. Note that `expected_improvement` is a static method, and also `fit_model` should be called from within one of the SMBO functions.

Note: While SMBO uses an internal surrogate model, please do not confuse it with the external surrogate model that we use to predict the performance of a given configuration (as ground truth).

Hint: Make use of plotting libraries to inspect the exact working of your surrogate function. If the surrogate function does not seem to work, you might either consider sampling more points, or intertwine the configurations that are generated by Bayesian optimization by random configurations (i.e., sample one configuration according to Bayesian optimization, sample one random, etc.), or likely both [1]. This should provide the surrogate model with a broad view of the input space.

2.2 Successive Halving

For this part of the assignment, you will again utilize the provided learning curve data to simulate an execution of successive halving. Successive halving is a resource allocation method that evaluates several configurations with limited resources and progressively allocates more resources to the top-performing model configurations.

Your task is to select datasets and proceed to simulate the successive halving based on the model's performance on progressively increasing anchor sizes. Similarly to earlier, each pair of machine learning algorithms and dataset should have an independently trained surrogate model; therefore one surrogate model per experimented dataset (machine learning algorithm fixed to *KNN*).

Consider a useful API, that is similar to the one provided for SMBO. We have not provided a template. Note that for this part of the functionality, you will need to use the 'anchor' functionality of the external surrogate (that can make predictions across different anchors).

Simulate the successive halving through a selection of best-performing model configurations, iteratively prune the poorly performing configurations (half at each round), and increase the anchor size.

3 Report and Submission

Report: Write a 3-page (maximum) report (excluding references) using the provided Latex template. Refrain from making changes to the template formatting. Demonstrate the working of both the surrogate model (e.g., using a holdout set and reporting the Spearman correlation), the SMBO algorithm optimizing *KNN* algorithm on several datasets using the surrogate model (for which we have provided learning curve data). Consider appropriate methods to plot the results and compare SMBO against Random Search and Successive Halving. Note that comparing SMBO against successive halving is not trivial, as they work on different scales.

Submission: Please provide the pdf (generated by your latex report) and the adjusted Python scripts. Ensure to hand in each Python script that produced a plot, so we can reproduce the plots of your report. Do not use packages that are hard to install (e.g., anything beyond Scikit-learn, scipy, numpy, pandas and ConfigSpace is typically not needed.) It is important that we can reproduce all plots that you created! **DO NOT** submit the pdf file in zip format since they need to be checked by our plagiarism tools, including *TurnItIn*.

References

[1] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.

[2] Felix Mohr, Tom J Viering, Marco Loog, and Jan N van Rijn. Lcdb 1.0: An extensive learning curves database for classification tasks. In *Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, volume 13717 of *Lecture Notes in Computer Science*, pages 3–19. Springer, 2022.

[3] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.