Sequential Model-based Optimization

AutoML – Assignment 1

Leiden University

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, 2]. SMBO can be written as follows, making use of 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))
```

In this assignment, we will finish these auxiliary functions. Note that this block of code is used in all unit tests. Please make sure to understand both the unit tests and the assignment files.

The assignment consists of three files,

- `assignment.py` – the file that needs to be filled in order to complete this assignment
- `test.py` – a file with unit tests

In the file `assignment.py`, you are expected to finish the following functions:

- `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 for each configuration what is the expected improvement.
- `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:

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

  where Φ and ϕ are the CDF and PDF of a standard normal distribution, respectively. (Note a slightly different formalation, 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 was found.

Note that for each function that needs to be implemented, you need to write approximately 3-10 lines of code.

Have a good look at the Unit tests. They specifically demonstrate how Sequential Model-based Optimization works and demonstrrate how the interface can be invoked. If all unit tests succeed, this is a good indication that the assignment could be completed successfully. Hint: make sure to work with the latest version of numpy (1.25.2), pandas (2.1.0), scikit-learn (1.3.0) and ConfigSpace (0.7.1) to prevent errors due to randomness.

Write a little report (2-pages) in Latex in which you demonstrate the working of the algorithm optimizing two different scikit-learn algorithms (e.g., Support Vector Machines and Adaboost) on several datasets from OpenML. You can use the scikit-learn function 'sklearn.datasets.fetch_openml()' for this. Motivate why you chose these datasets. Think about an appropriate way to plot the results using matplotlib. Compare it against Grid Search and Random Search (as implemented in Scikit-Learn) before the deadline.

Hand in the files `assignment.py`, as well as the report in PDF format.

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] 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.