



AutoML for Bioinformatics

Johann Dréo

► To cite this version:

Johann Dréo. AutoML for Bioinformatics. Doctoral. Computational Biology Seminars, France. 2022, pp.31. hal-04125770

HAL Id: hal-04125770

<https://hal.science/hal-04125770>

Submitted on 12 Jun 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution - ShareAlike 4.0 International License

AutoML for Bioinformatics

• Johann Dreo • 2022-06-21



Abstract

Computational Intelligence has achieved considerable successes in recent years and numerous applications rely on it. However, its use crucially relies on human experts, from feature engineering to algorithm selection. Since the performance of a given approach depends on the qualities of the algorithm regarding the targeted data and task, the design of new application becomes more and more difficult. Benchmarking, algorithm selection and hyperparameter tuning thus become tools of choice for helping non-experts to rigorously design the method that will solve their problem. This presentation aims at introducing the (relatively) new scientific field arising around the progressive automation of computational intelligence.

Summary

01

—

Automated Design
Problem

02

—

Solutions

03

—

Perspectives

04

—

Hands on



Part 1

Automated Design Problem

Context

Computational Intelligence

Artificial Intelligence:

- Computational Intelligence
 - Neural Networks
 - Evolutionary Computing
 - Fuzzy Logic

Context

e.g. Machine Learning

Tasks:

- Data Management
- Design of Features
- Selection of Model
- Model Parameters Tuning
- Design Topology of NN
- Postprocess Models
- Analyze Results

Demands for off-the-shelf, automated, tools.

Why?

Simple example in bioinformatics

- Olson *et al.*, *Data-driven advice for applying machine learning to bioinformatics problems*, Pacific Symposium on Biocomputing 2018
- 13 scikit-learn algorithms
- 165 bioinfo classification problems
- model selection & hyperparameters tuning
- with a grid « search »

Why?

Performances

- Improvement in accuracy compared to the average on each dataset.
- 20% av. increase in accuracy
- up to more than 60%

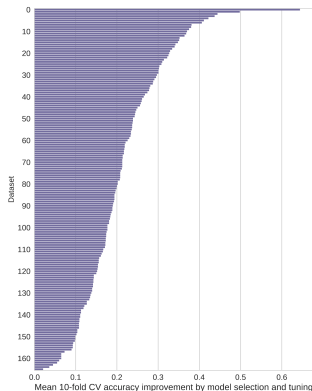


Fig. 4. Improvement in 10-fold CV accuracy by model selection and tuning, relative to the average performance on each dataset.

Main problem

Optimization

Design, Selection, Tuning = *Optimization*

Main problem

Optimization

Design, Selection, Tuning = *Optimization*

Given a random set of problem instances $I \sim \mathcal{I}$ drawn in all the possible problems $\mathcal{I} = \mathcal{P}(\mathcal{J})$, the algorithm selection problem is to find the mapping from problems to algorithms $f : I \mapsto A$ that maximize a measure of central tendency of the performance metric distribution

Main problem

Optimization

Design, Selection, Tuning = *Optimization*

Given a random set of problem instances $I \sim \mathcal{I}$ drawn in all the possible problems $\mathcal{I} = \mathcal{P}(\mathcal{J})$, the algorithm selection problem is to find the mapping from problems to algorithms $f : I \mapsto A$ that maximize a measure of central tendency of the performance metric distribution:

$$f(I) = \arg \max_{A \in \mathcal{A}} \mathbb{E}_{i \in I} p(i, A(i))$$

Main problem

Optimization

Design, Selection, Tuning = *Optimization*

Given a random set of problem instances $I \sim \mathcal{I}$ drawn in all the possible problems $\mathcal{I} = \mathcal{P}(\mathcal{J})$, the algorithm selection problem is to find the mapping from problems to algorithms $f : I \mapsto A$ that maximize a measure of central tendency of the performance metric distribution:

$$f(I) = \arg \max_{A \in \mathcal{A}} \mathbb{E}_{i \in I} p(i, A(i))$$

Key difference: representation of A

Main problem

Optimization

Design, Selection, Tuning = *Optimization*

Given a random set of problem instances $I \sim \mathcal{I}$ drawn in all the possible problems $\mathcal{I} = \mathcal{P}(\mathcal{J})$, the algorithm selection problem is to find the mapping from problems to algorithms $f : I \mapsto A$ that maximize a measure of central tendency of the performance metric distribution:

$$f(I) = \arg \max_{A \in \mathcal{A}} \mathbb{E}_{i \in I} p(i, A(i))$$

Key difference: representation of A

Key feature: p is non-differentiable

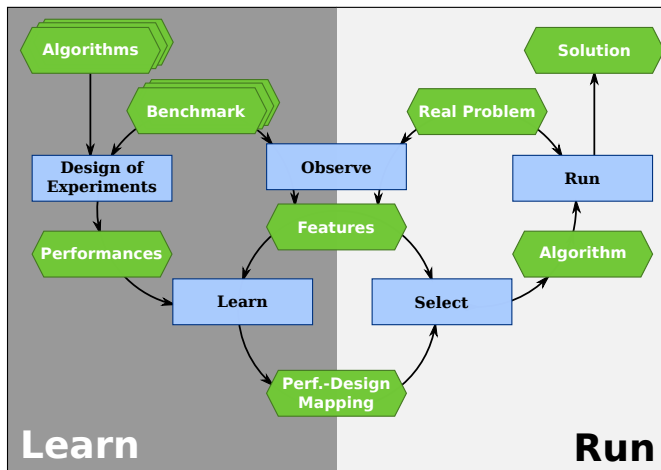
Main problem

Sub-problems

- Hyperparameter Optimization / Algorithm Configuration
- Algorithm Selection
- Neural Architecture Search
- Algorithm Design
- Meta-Learning / Dynamic Algorithm Configuration / Learning to Learn

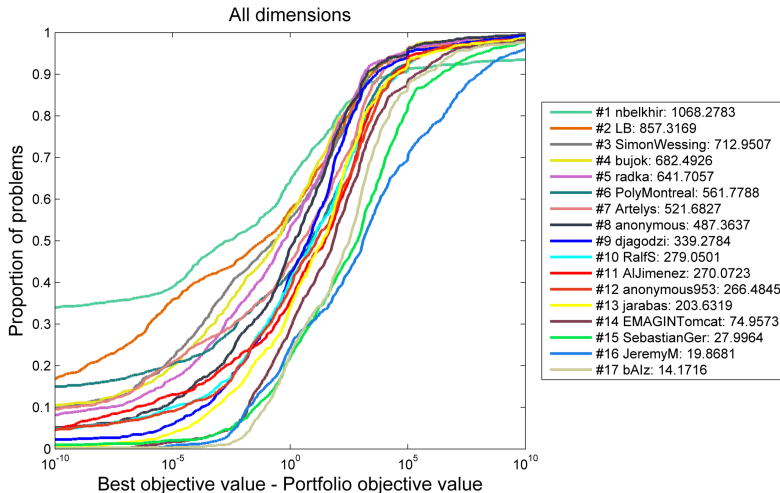
Meta-Learning example

Landscape-aware hyperparameters tuning



Meta-Learning example

Landscape-aware hyperparameters tuning



Part 2

Solutions



Algorithmics

Two main families:

- Bayesian Optimization
- Evolutionary Algorithms

Common issue: how to assess performance

- Reproducible Benchmarking
- Cross-validation

Algorithmics

Randomized Search

Common features:

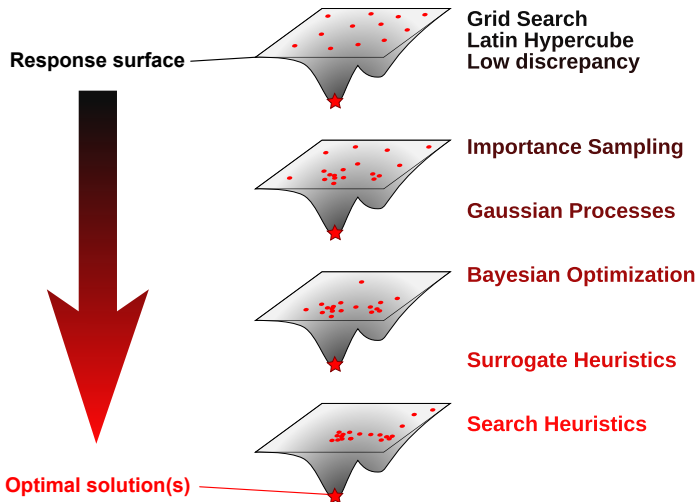
- Sample the search space
- Bias toward bests areas
- Iteratively

Bayesian:

- Manage the randomization of the problem
- Compromise perf/uncertainty

Algorithmics

A gradient



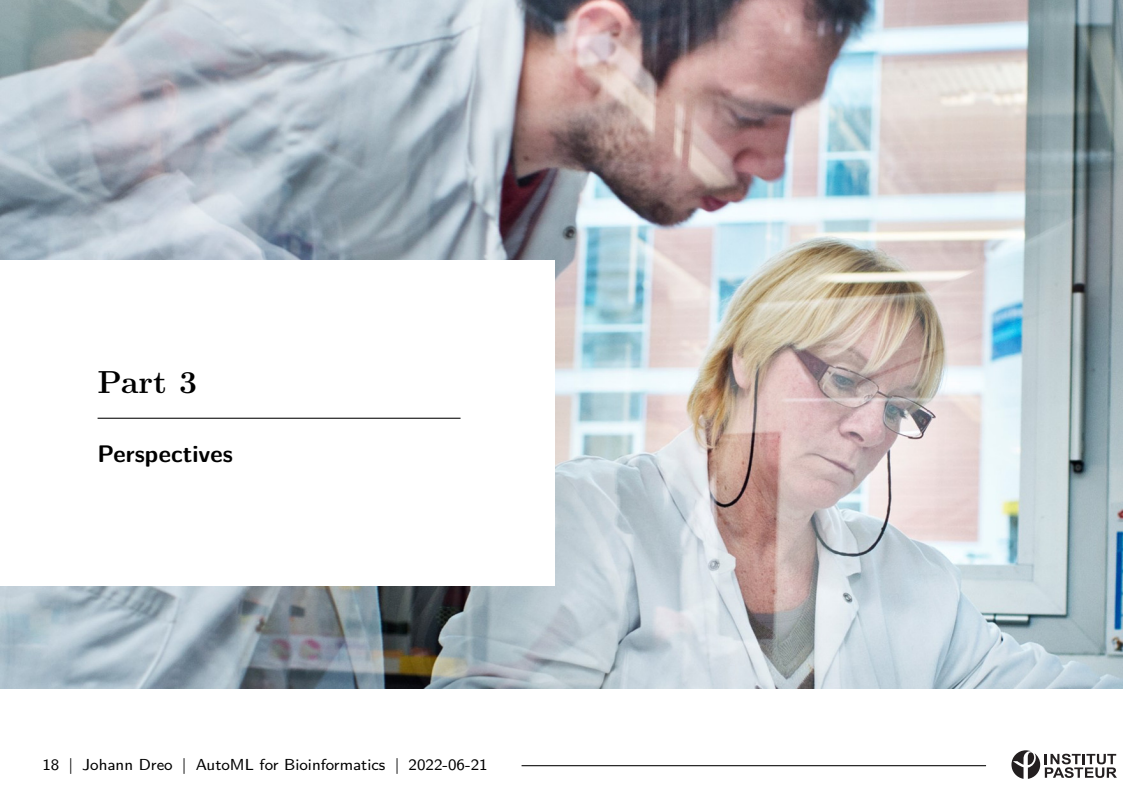
Algorithmics

Software Tools

Generic libraries:

- AutoWEKA (selection and tuning), Auto-sklearn
- Auto-PyTorch (tuning and archi)
- AutoGluon (tuning, model selection, archi)
- H2O AutoML (model selection)
- MLBox (feature selection, tuning)
- TPOT (pipeline optimization)
- Nevergrad (benchmarking, tuning, selection)

+ tons of dedicated solvers.



Part 3

Perspectives

Pareto analysis

- Search for trade-offs
- Between various performances metrics
- Another kind of meta-learning
- Allow the user to have preferences

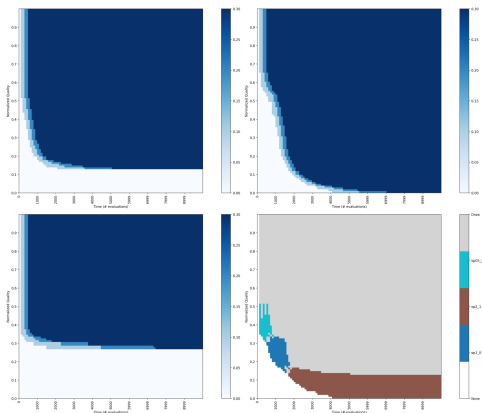


Figure 2: Dominance map for three instances CMA-ES on the BBOB Katsuura Lunacek Bi-Rastrigin function, dimension 50, first instance, with different population sizes (upper-left/blue: default population size, lower-left/cyan: 1/2 times the default, upper-right/brown: 2 times the default). White area shows the domain never attained by any algorithm, while gray area shows the domain attained by at least two algorithms with the same probability and colored area the domain where a single algorithm attain the higher probability.

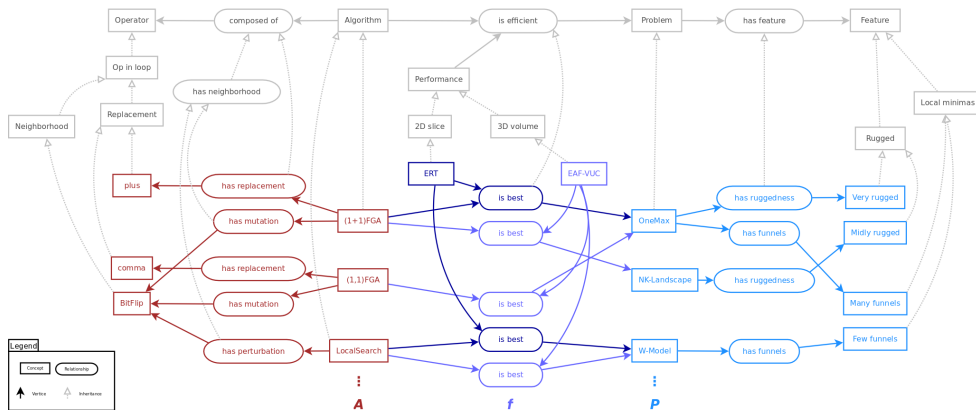
Interactive / Explainable

- Parameters importance / Automatic Ablation Studies
- Visualizations
- Human-guided search
- Automated integration of results in the OPTION Ontology

Interactive / Explainable

Data integration in OPTION: why?

- Backup
- Reproducibility
- Help algorithmicians



Engineering

- So far, in the ascending phase
- Lot (too many) of options to choose from
- Which platform will win?

Engineering

Link

- <https://www.automl.org>

Part 4

Hands on



What's needed

- Type of objective: mono-objective
- Type of function: noisy
- Problem generator:
 - Several [ML] models
 - Several hyper-parameters
 - Automated call to single runs (e.g. cross-validation)
 - Probably a working SLURM-compatible pipeline
- Type of parameters: continuous, qualitative or mixed?

Elpida

How to expose the problem to a solver?

- (Advertisement) Elpida protocol: <https://github.com/jdreo/elpida>
- Messages exchange: Problem server \iff Solver client
- Simple JSON messages
- Read files

Elpida

JSON messages

You receive:

```
{  
    "query_type": "call",  
    "solution": [10,10]  
}
```

You answer:

```
{  
    "query_type": "value",  
    "value": [3.14]  
}
```

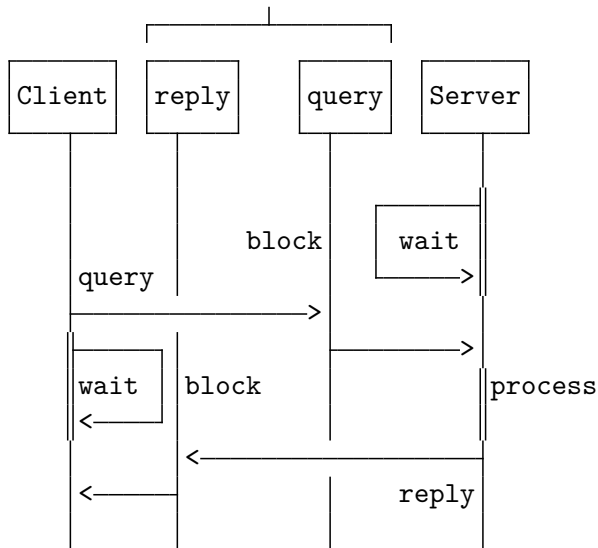

Read files

```
import json
# Wait for a query:
with open("query",'r') as fd:
    squery = fd.read()
# Parse the message:
jqquery = json.loads(squery)
# Extract the solution:
sol = jqquery["solution"]
# Compute the value of the objective function:
val = sum(sol)
# Forge a JSON `value` reply:
sreply = json.dumps( { "query_type":"value", "value":[val] } )
# Send the reply:
with open("reply",'w') as fd:
    fd.write(sreply)
```

(in a loop)

Elpida

FIFO named pipes



Elpida

Example: plugging SMAC3

Using `elpida/examples/smac3/smac3-elpida-cli.py`:

```
parser.add_argument("-x", metavar="X", type=float,  
                    help="A variable")  
parser.add_argument("-y", metavar="Y", type=float,  
                    help="Another variable")
```

Test a single run:

```
../python/problem_server.py # Run problem server  
../smac3-elpida-cli.py 0 1 2 3 4 -x 2 -y 3 # One design
```

Running SMAC3

scenario.txt:

```
algo = smac3-elpida-cli.py
paramfile = configspace.pcs
run_obj = quality
runcount_limit = 10
deterministic = 1
```

configspace.pcs:

```
# name [min,max] [default]
x [-5,10] [0]
y [0,15] [0]
```

Run SMAC:

```
smac.py --scenario scenario.txt
```

Thanks for your attention!

And don't worry, I will help you.