

AutoML: Beyond AutoML

Best Practices for Algorithm Configuration

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Pitfalls & Best Practices

The **goals** of automated algorithm configuration include:

Pitfalls & Best Practices

The **goals** of automated algorithm configuration include:

- 1 reducing the expertise required to use an algorithm

Pitfalls & Best Practices

The **goals** of automated algorithm configuration include:

- 1 reducing the expertise required to use an algorithm
- 2 less human-time

Pitfalls & Best Practices

The **goals** of automated algorithm configuration include:

- ➊ reducing the expertise required to use an algorithm
- ➋ less human-time
- ➌ tuning algorithms to the task at hand

Pitfalls & Best Practices

The **goals** of automated algorithm configuration include:

- ➊ reducing the expertise required to use an algorithm
- ➋ less human-time
- ➌ tuning algorithms to the task at hand
- ➍ faster development of algorithms

Pitfalls & Best Practices

The **goals** of automated algorithm configuration include:

- ➊ reducing the expertise required to use an algorithm
- ➋ less human-time
- ➌ tuning algorithms to the task at hand
- ➍ faster development of algorithms
- ➎ facilitating systematic and reproducible research

BUT:

- ➊ algorithm configuration can lead to over-tuning

Pitfalls & Best Practices

The **goals** of automated algorithm configuration include:

- ➊ reducing the expertise required to use an algorithm
- ➋ less human-time
- ➌ tuning algorithms to the task at hand
- ➍ faster development of algorithms
- ➎ facilitating systematic and reproducible research

BUT:

- ➊ algorithm configuration can lead to over-tuning
- ➋ using algorithm configuration requires (at least some) expertise in algorithm configuration

Pitfalls & Best Practices

The **goals** of automated algorithm configuration include:

- ➊ reducing the expertise required to use an algorithm
- ➋ less human-time
- ➌ tuning algorithms to the task at hand
- ➍ faster development of algorithms
- ➎ facilitating systematic and reproducible research

BUT:

- ➊ algorithm configuration can lead to over-tuning
- ➋ using algorithm configuration requires (at least some) expertise in algorithm configuration
- ➌ if done wrong, waste of time and compute resources

Setting up AC

9 Steps to your well-performing algorithm:

- 1 Define your performance metric

Setting up AC

9 Steps to your well-performing algorithm:

- 1 Define your performance metric
- 2 Define instance set

Setting up AC

9 Steps to your well-performing algorithm:

- 1 Define your performance metric
- 2 Define instance set
- 3 Split your instances in training and test

Setting up AC

9 Steps to your well-performing algorithm:

- 1 Define your performance metric
- 2 Define instance set
- 3 Split your instances in training and test
- 4 Define the configuration space

Setting up AC

9 Steps to your well-performing algorithm:

- 1 Define your performance metric
- 2 Define instance set
- 3 Split your instances in training and test
- 4 Define the configuration space
- 5 Choose your preferred configurator

Setting up AC

9 Steps to your well-performing algorithm:

- 1 Define your performance metric
- 2 Define instance set
- 3 Split your instances in training and test
- 4 Define the configuration space
- 5 Choose your preferred configurator
- 6 Implement an interface between your algorithm and the configurator

Setting up AC

9 Steps to your well-performing algorithm:

- 1 Define your performance metric
- 2 Define instance set
- 3 Split your instances in training and test
- 4 Define the configuration space
- 5 Choose your preferred configurator
- 6 Implement an interface between your algorithm and the configurator
- 7 Define resource limitations of your algorithm

Setting up AC

9 Steps to your well-performing algorithm:

- 1 Define your performance metric
- 2 Define instance set
- 3 Split your instances in training and test
- 4 Define the configuration space
- 5 Choose your preferred configurator
- 6 Implement an interface between your algorithm and the configurator
- 7 Define resource limitations of your algorithm
- 8 Run the configurator on your algorithm and the training instances

Setting up AC

9 Steps to your well-performing algorithm:

- 1 Define your performance metric
- 2 Define instance set
- 3 Split your instances in training and test
- 4 Define the configuration space
- 5 Choose your preferred configurator
- 6 Implement an interface between your algorithm and the configurator
- 7 Define resource limitations of your algorithm
- 8 Run the configurator on your algorithm and the training instances
- 9 Validate the eventually returned configuration on your test instances

Pitfall 1: Trust your algorithm

We have encountered algorithms that

- ignored resource limitations
- returned wrong solutions
- even returned negative runtimes

Pitfall 1: Trust your algorithm

We have encountered algorithms that

- ignored resource limitations
- returned wrong solutions
- even returned negative runtimes

Best Practice 1: Never trust your algorithm

Explicitly check and use external software to:

- ① ensure resource limitations
- ② terminate your algorithm
- ③ verify returned solutions
- ④ measure performance

Pitfall 2: File System

Algorithm configurators ...

- often produce quite some log files (e.g., for each algorithm run)
- are often used on HPC clusters with a shared file system

Pitfall 2: File System

Algorithm configurators ...

- often produce quite some log files (e.g., for each algorithm run)
- are often used on HPC clusters with a shared file system

⇒ It's surprisingly easy to substantially slow down a shared file system

Pitfall 2: File System

Algorithm configurators ...

- often produce quite some log files (e.g., for each algorithm run)
- are often used on HPC clusters with a shared file system

⇒ It's surprisingly easy to substantially slow down a shared file system

Best Practice 2: Don't use the Shared File System

To relieve the file system of a HPC cluster:

- design well which files are required and which are not
- use a local (SSD) disc

Pitfall 3: Over-tuning

It's easy to overtune to different aspects, incl.:

- training instances
- random seeds
- machine type

Pitfall 3: Over-tuning

It's easy to overtune to different aspects, incl.:

- training instances
- random seeds
- machine type

In practice, it can be hard to prevent over-tuning, e.g., by

- using larger instance sets
- tuning on the target hardware

Best Practice 3: Check for Over-Tuning

Check for over-tuning by validating your final configuration on

- many random seeds
- a large set of unused test instances
- a different hardware

Pitfall 4: Heterogeneous Instance Sets

Algorithm configurators...

- use some kind of racing to not evaluate each configuration on all instances

Pitfall 4: Heterogeneous Instance Sets

Algorithm configurators...

- use some kind of racing to not evaluate each configuration on all instances
- can be mislead on subsets of instances if the instance set is heterogeneous

Pitfall 4: Heterogeneous Instance Sets

Algorithm configurators...

- use some kind of racing to not evaluate each configuration on all instances
- can be mislead on subsets of instances if the instance set is heterogeneous

~> returned configurations often perform worse than default configurations
in the validation phase

Pitfall 4: Heterogeneous Instance Sets

Algorithm configurators...

- use some kind of racing to not evaluate each configuration on all instances
- can be mislead on subsets of instances if the instance set is heterogeneous

⇒ returned configurations often perform worse than default configurations in the validation phase

Best Practice 4: Ensure Homogeneity

Algorithm configurators should only run on homogeneous instance sets.

Different degrees of homogeneity:

- Strong homogeneity: all instances agree on the ranking of configurations
- Weak homogeneity: all instances agree on the top-performing configurations

More Pitfalls and Best Practices

... can be found in [Eggensperger et al. 2019]