Automated Machine Learning

DS&AI Conference, 4 October 2022, Bangkok

Matthias König

Universiteit Leiden

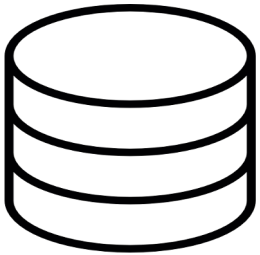
This slide deck was prepared together with Mike Huisman

Automated Machine Learning

Automated Machine Learning

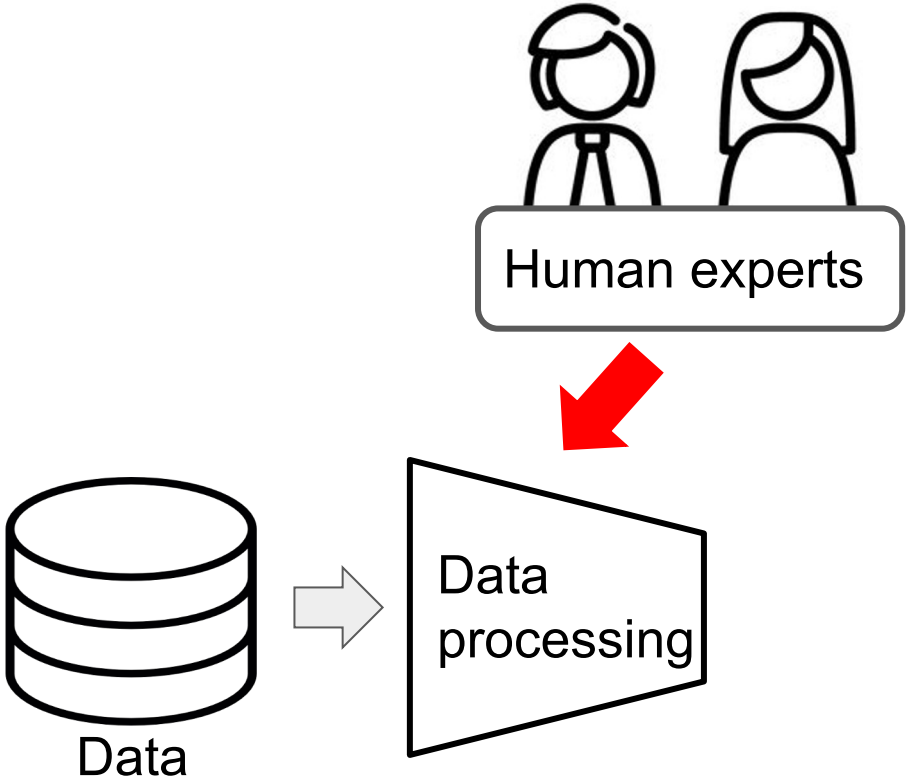
Machine learns to perform task(s) from data

Automated Machine Learning

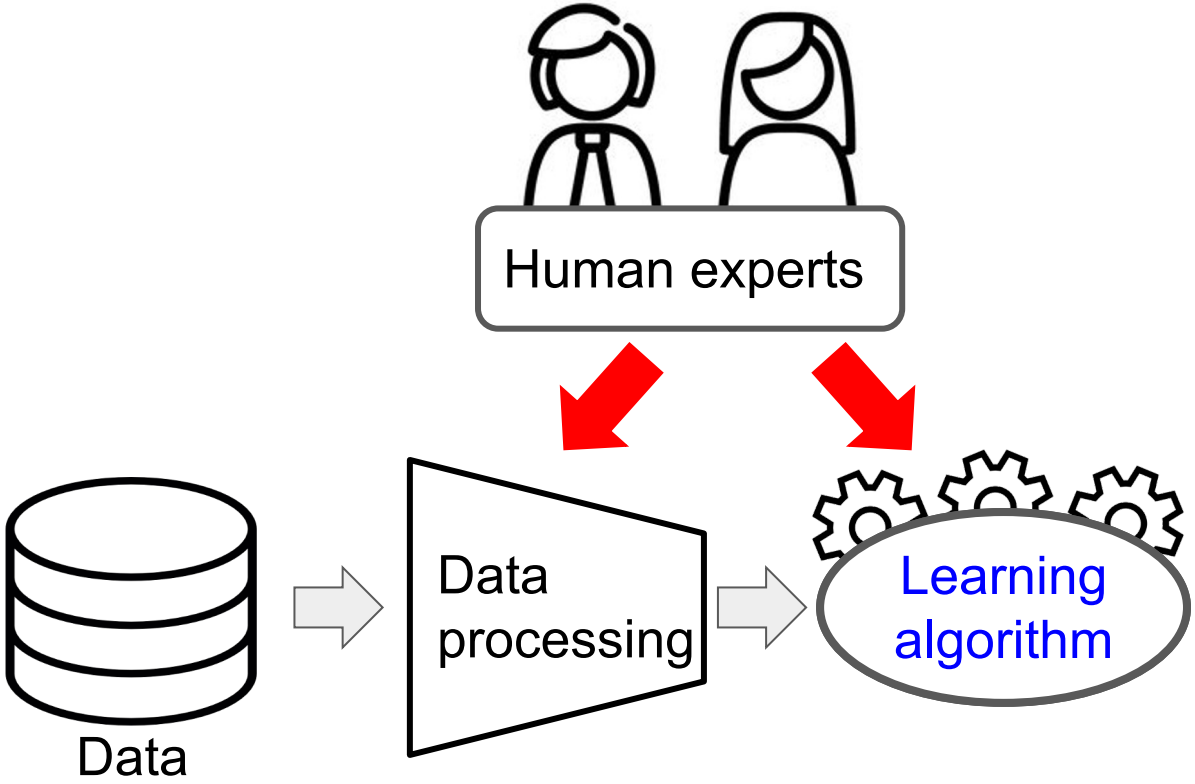


Data

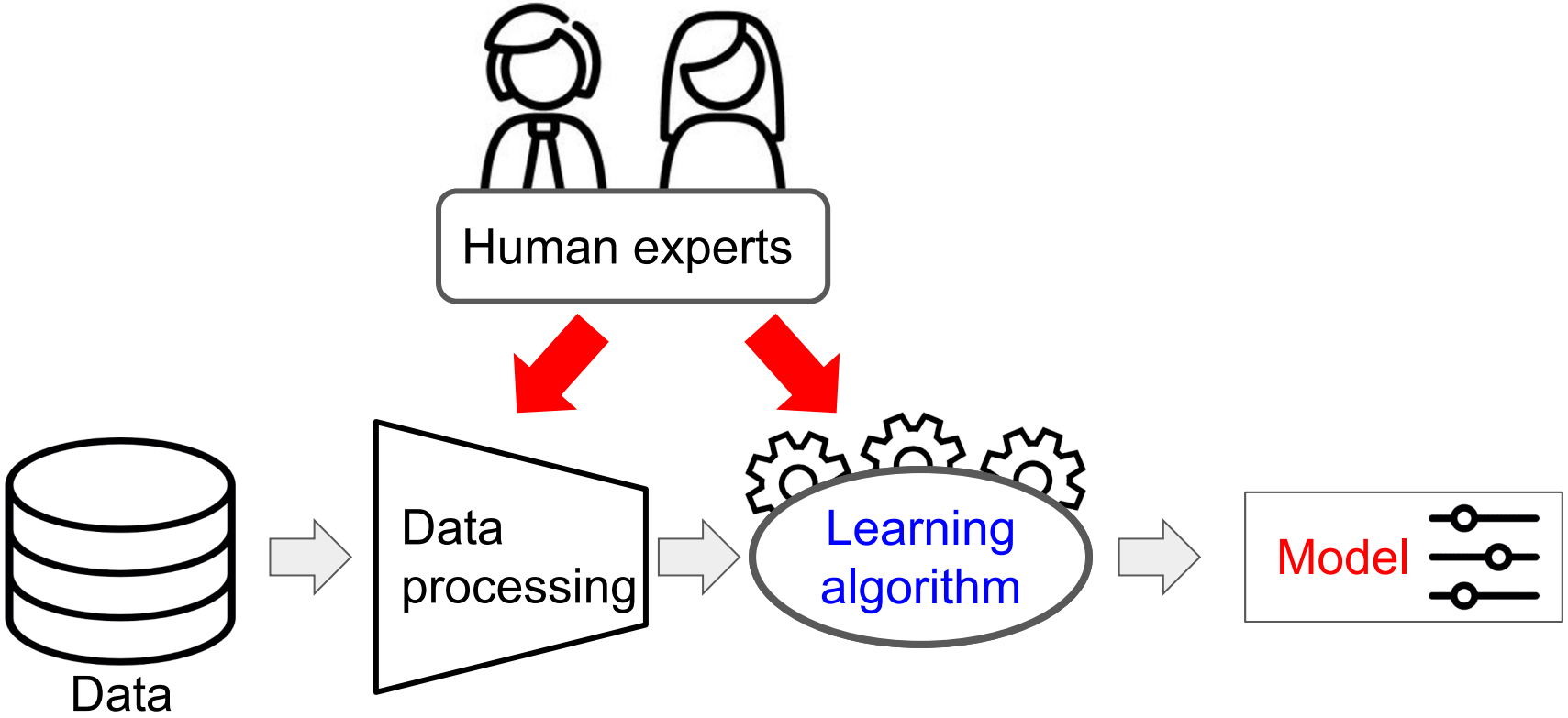
Automated Machine Learning



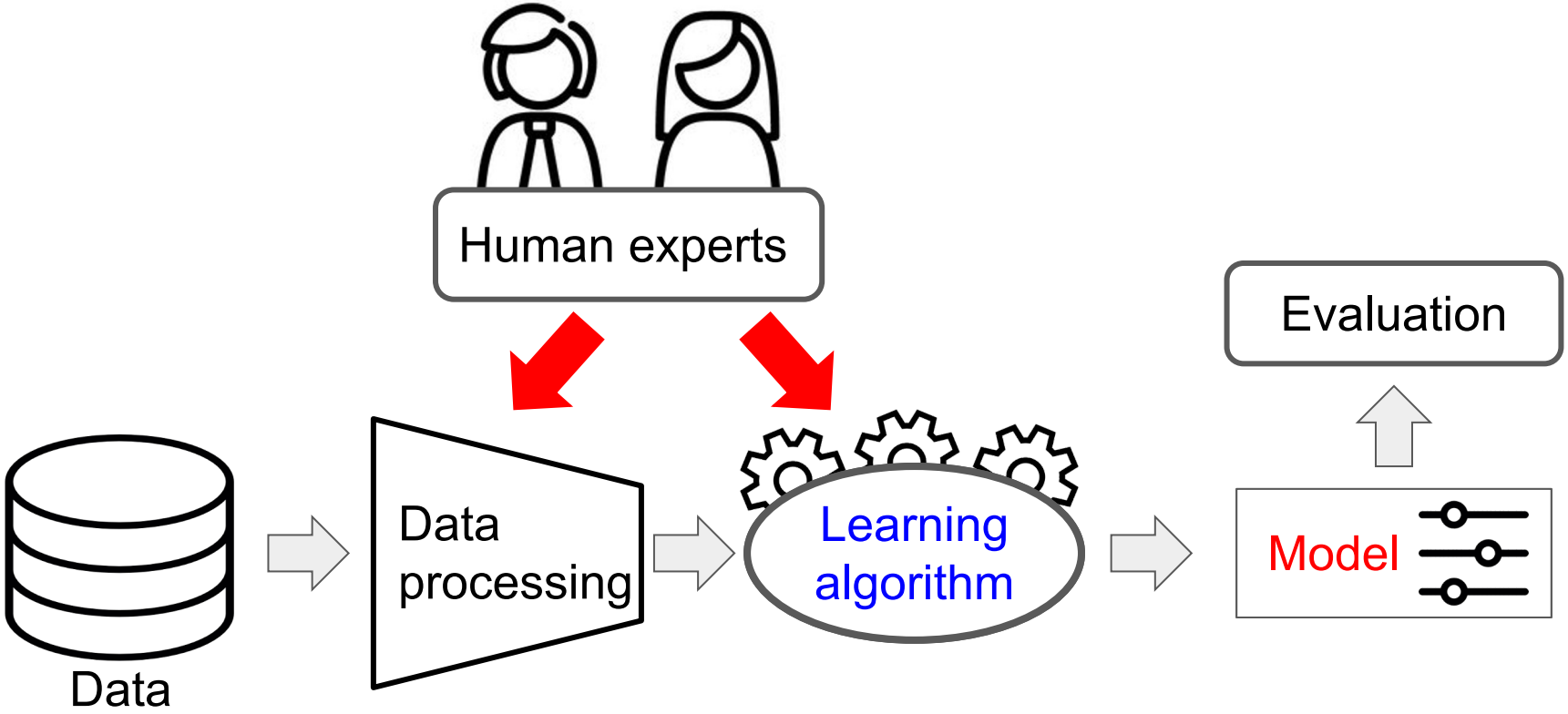
Automated Machine Learning



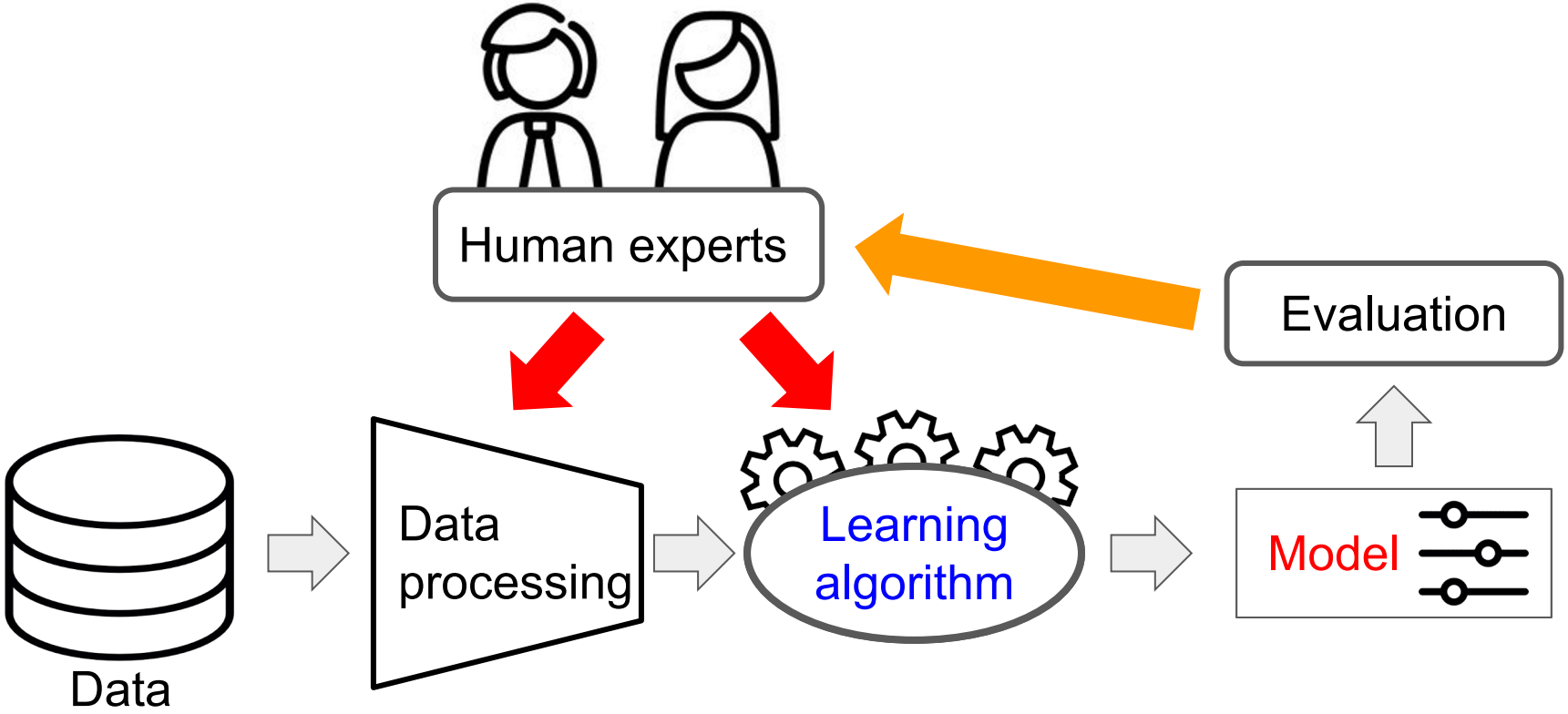
Automated Machine Learning



Automated Machine Learning

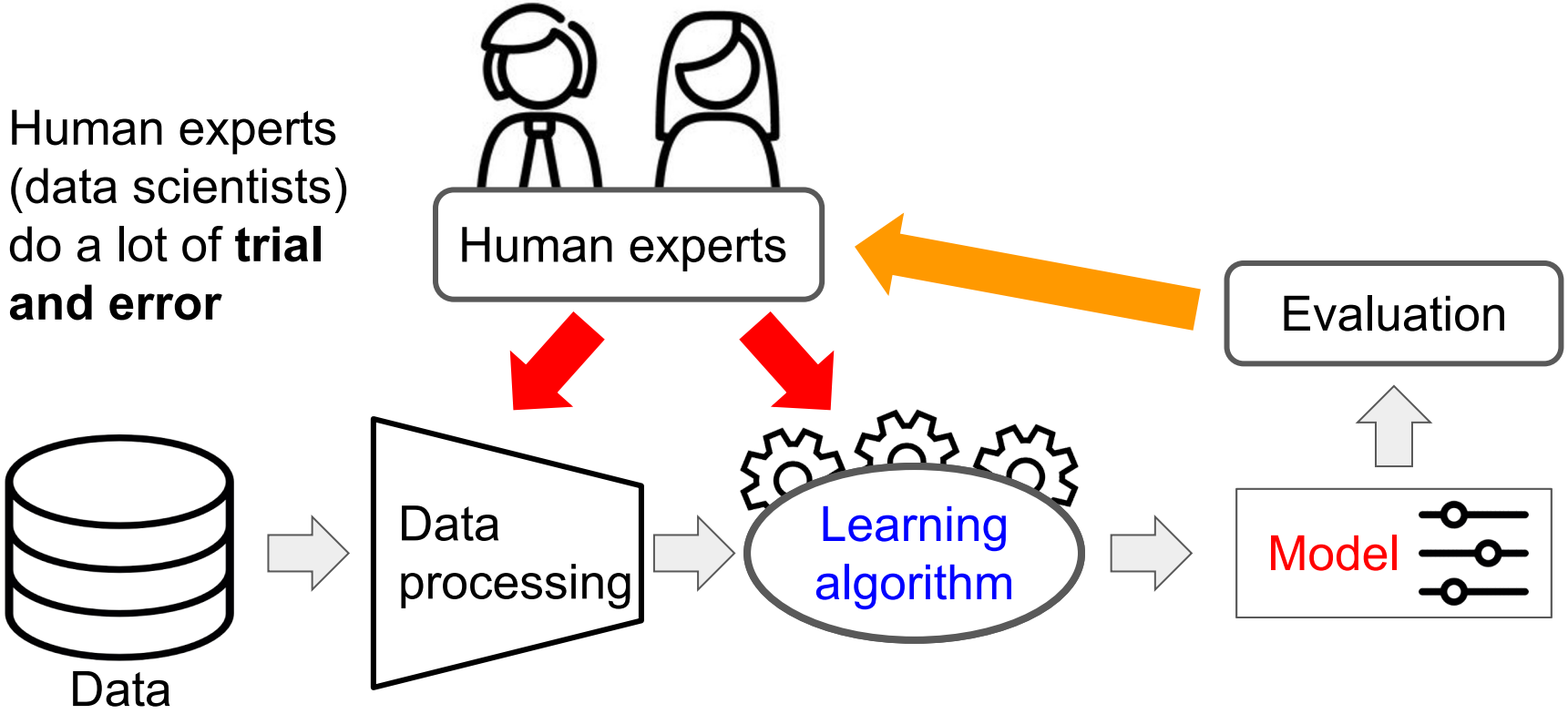


Automated Machine Learning

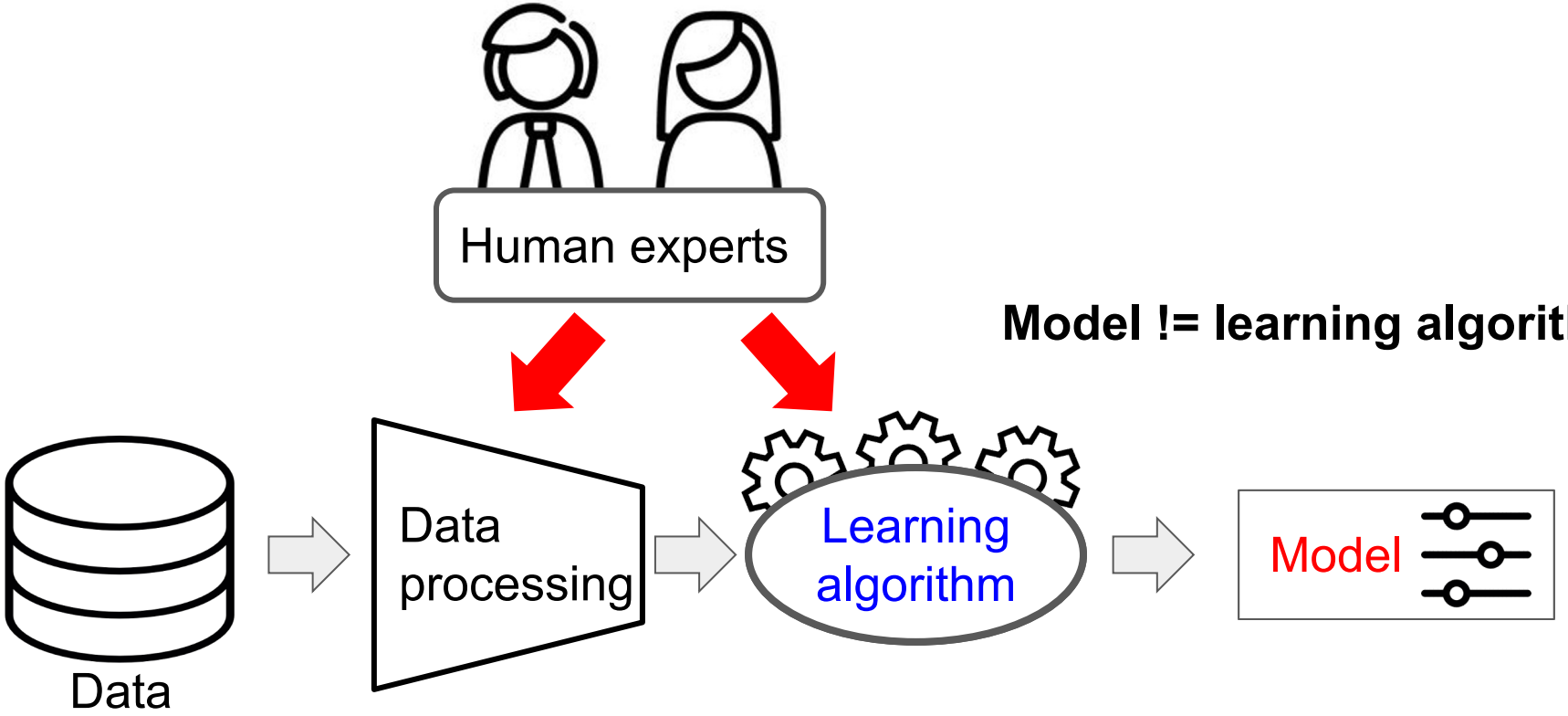


Automated Machine Learning

Human experts (data scientists) do a lot of **trial and error**

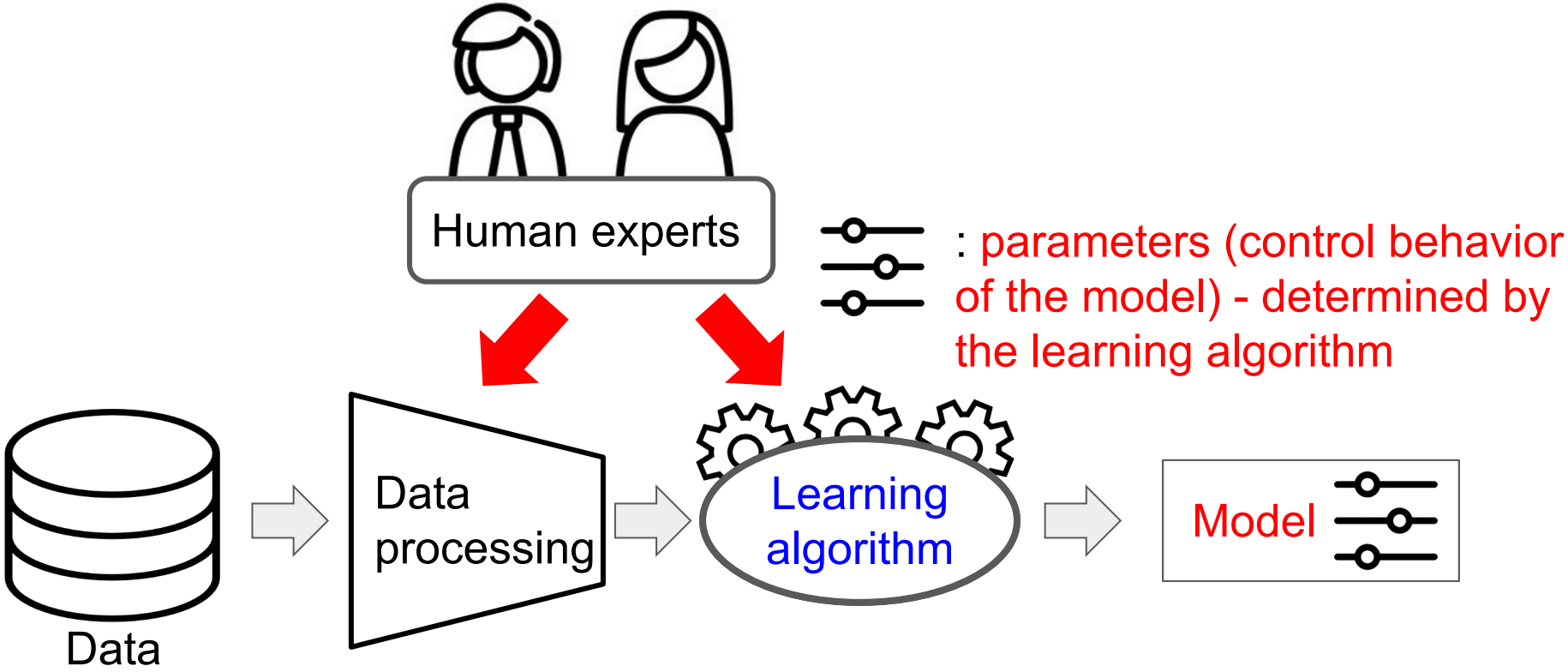


Automated Machine Learning

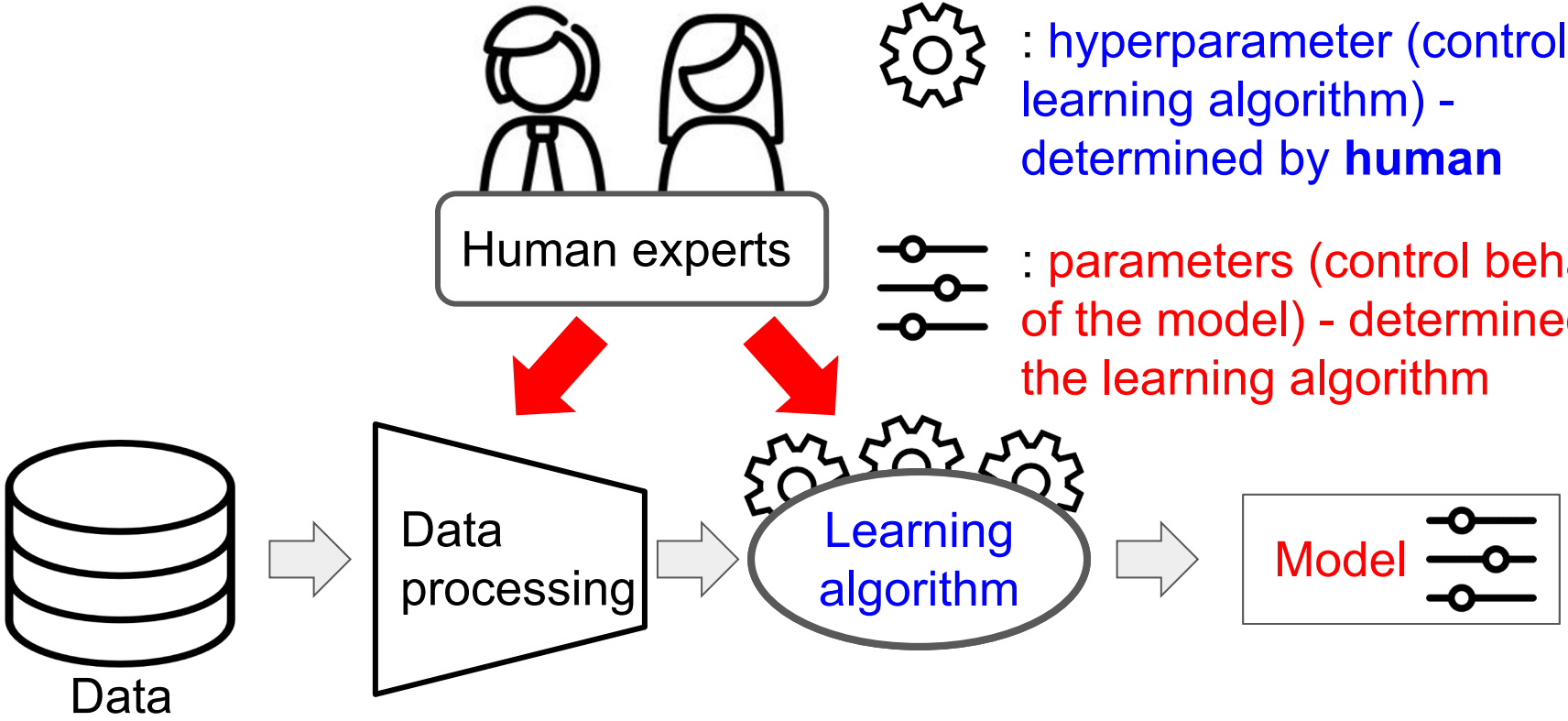


Model != learning algorithm

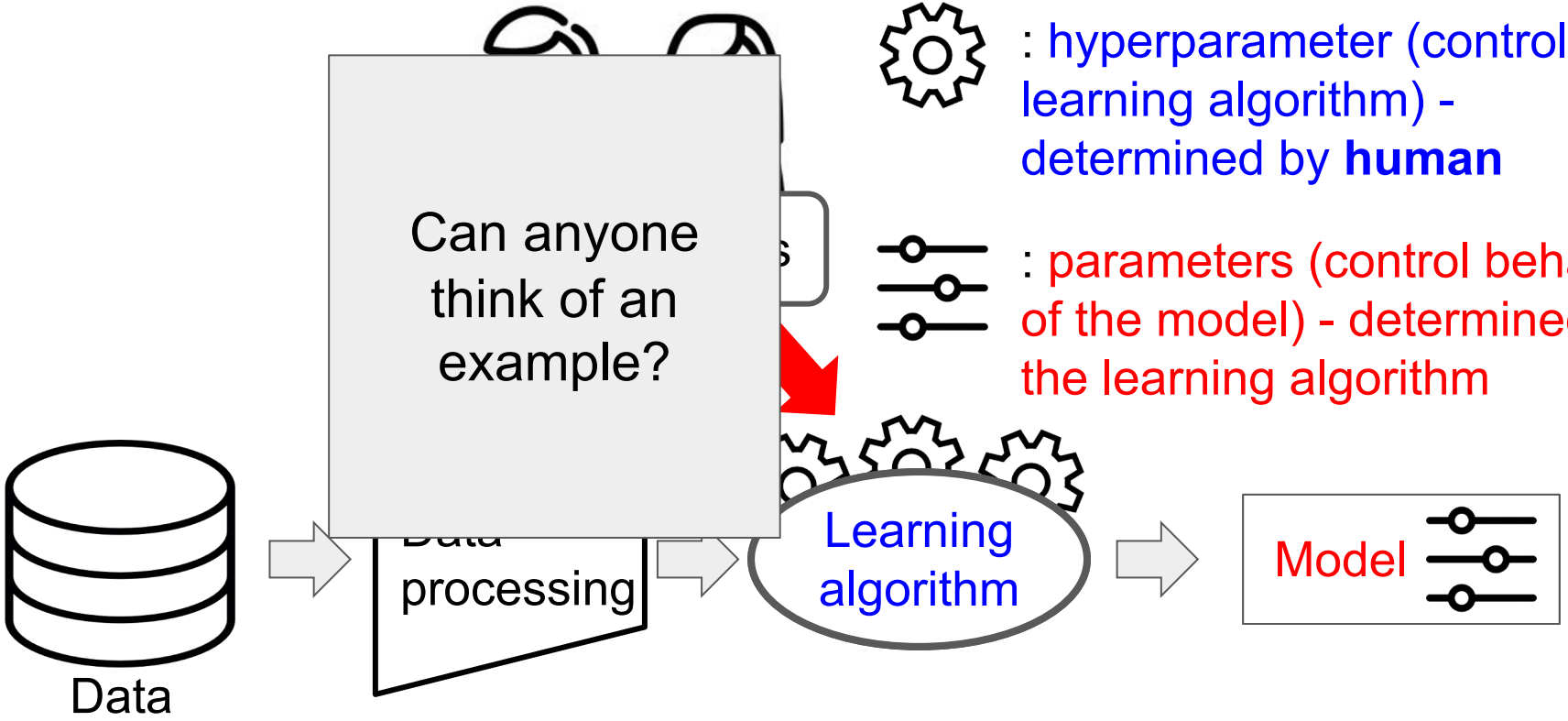
Automated Machine Learning



Automated Machine Learning



Automated Machine Learning



Role of the human experts

- What data processing steps (cleaning, feature engineering, etc.) do we use?

Role of the human experts

- What data processing steps (cleaning, feature engineering, etc.) do we use?
- What learning algorithm do we use? (SVM, Logistic regression, Neural Network,...) = **Algorithm selection**

Role of the human experts

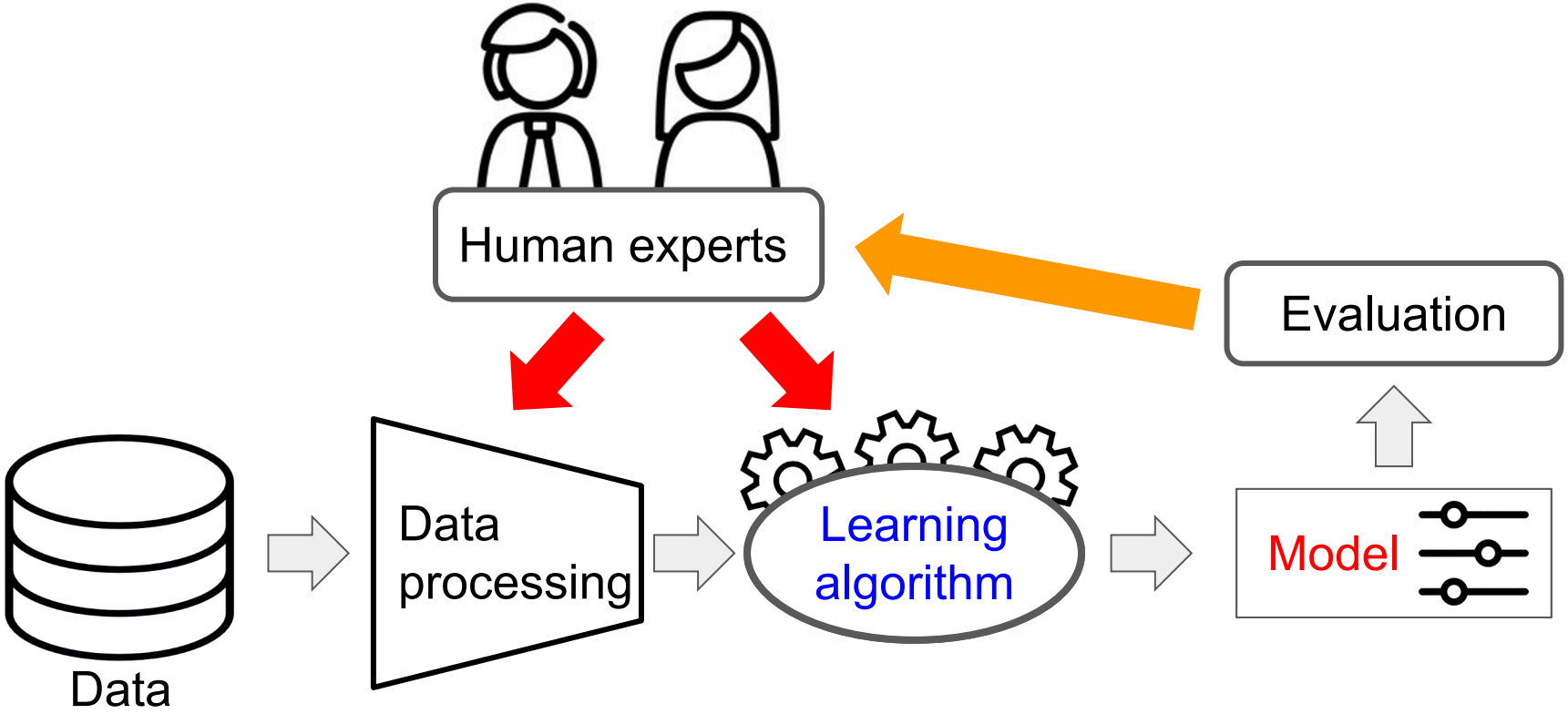
- What data processing steps (cleaning, feature engineering, etc.) do we use?
- What learning algorithm do we use? (SVM, Logistic regression, Neural Network,...) = **Algorithm selection**
- What **hyper**parameters do we use for the given learning algorithms? **= Hyperparameter selection**

Role of the human experts

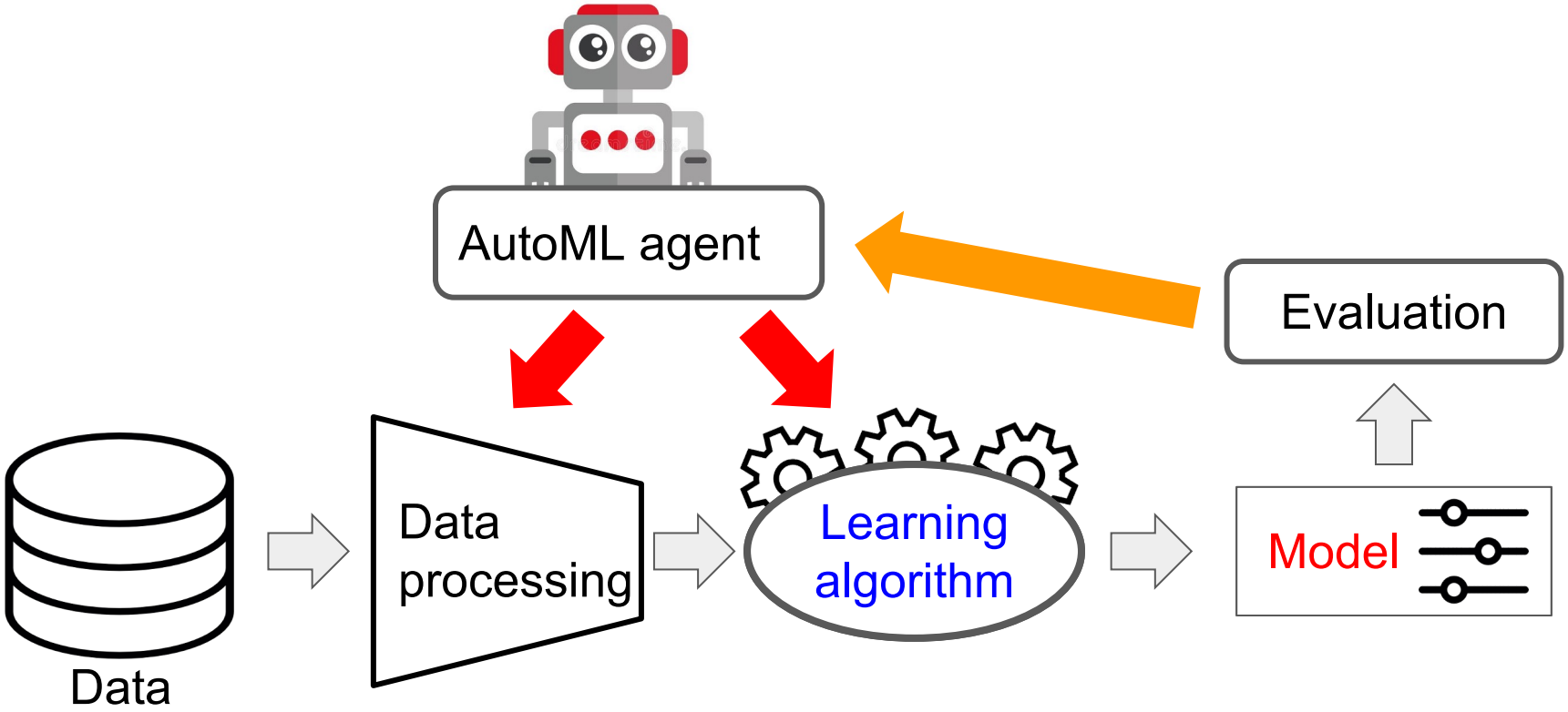
- What data processing steps (cleaning, feature engineering, etc.) do we use?
- What learning algorithm do we use? (SVM, Logistic regression, Neural Network,...) = **Algorithm selection**
- What **hyper**parameters do we use for the given learning algorithms? **= Hyperparameter selection**

Automated Machine learning = Automating one or more of these things

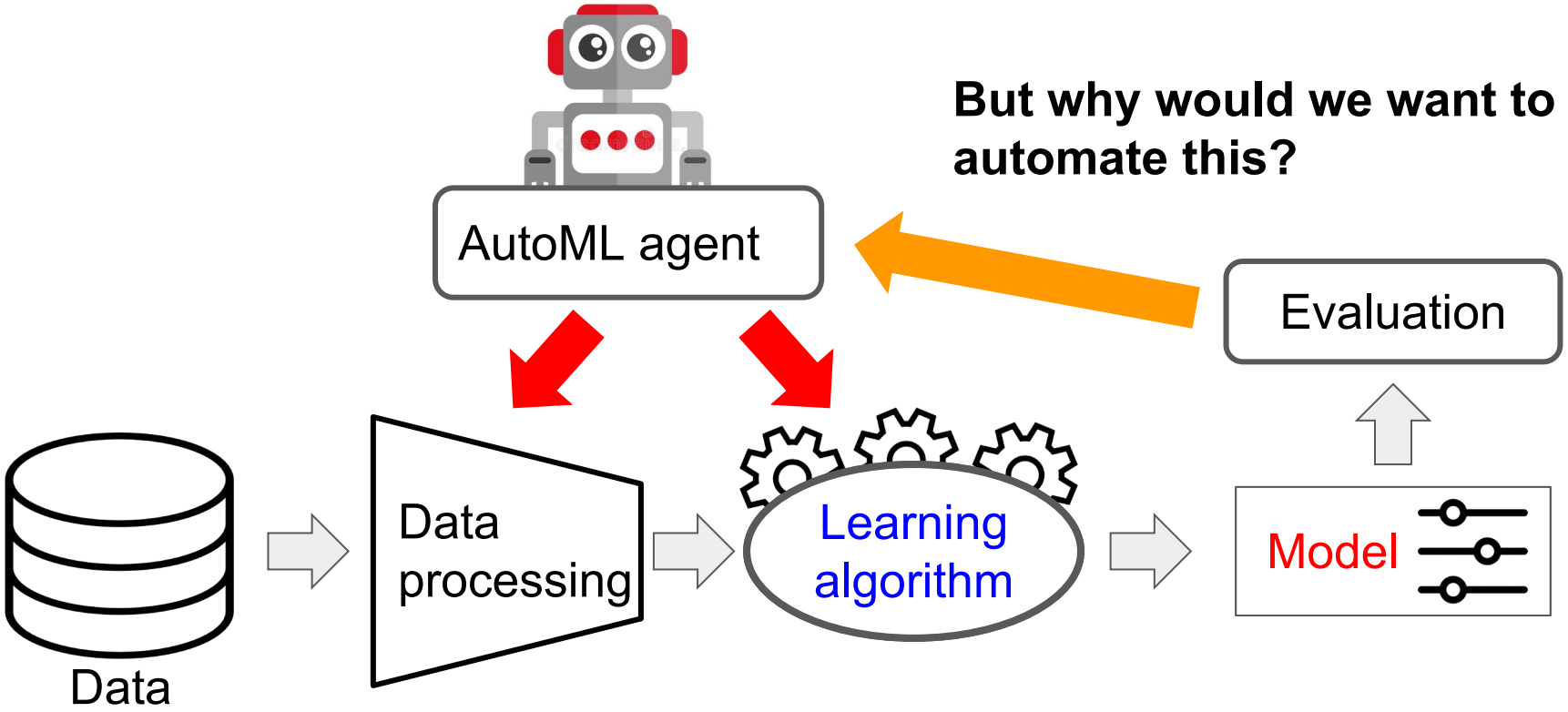
Automated Machine Learning



Automated Machine Learning



Automated Machine Learning



Why automate ML?

Choice of preprocessing, learning algorithm, and hyperparameters is **CRUCIAL** for achieving good performance!

Why automate ML?

Choice of preprocessing, learning algorithm, and hyperparameters is **CRUCIAL** for achieving good performance!

But finding good settings for these components:

Why automate ML?

Choice of preprocessing, learning algorithm, and hyperparameters is **CRUCIAL** for achieving good performance!

But finding good settings for these components:

- Requires expert domain knowledge (limits widespread use ML)

Why automate ML?

Choice of preprocessing, learning algorithm, and hyperparameters is **CRUCIAL** for achieving good performance!

But finding good settings for these components:

- Requires expert domain knowledge (limits widespread use ML)
- Takes **a lot of (tedious) effort** and is **expensive** (read this logbook for the creation of a 175B parameter model)

Why automate ML?

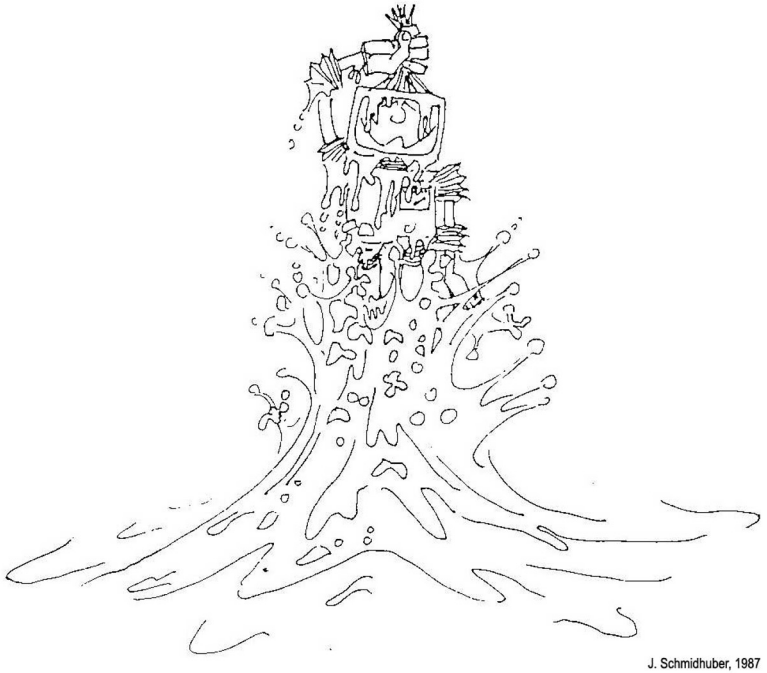
Choice of preprocessing, learning algorithm, and hyperparameters is **CRUCIAL** for achieving good performance!

But finding good settings for these components:

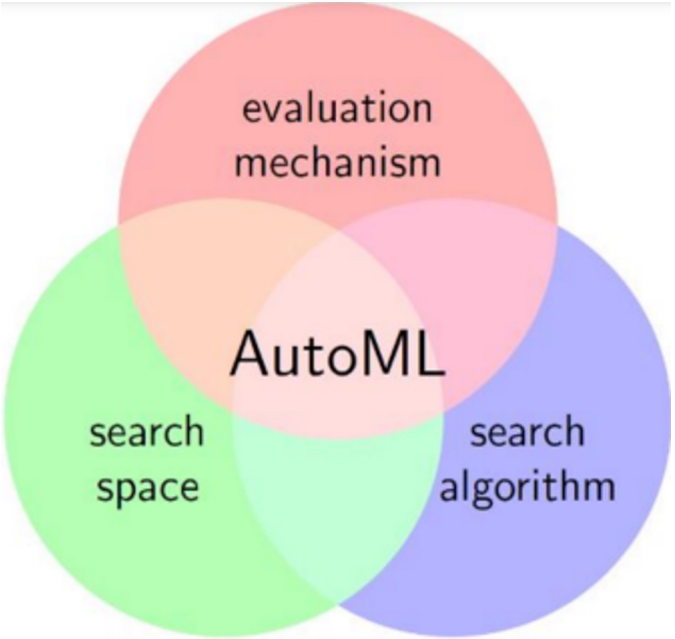
- Requires expert domain knowledge (limits widespread use ML)
- Takes **a lot of (tedious) effort** and is **expensive** (read this logbook for the creation of a 175B parameter model)
- Suboptimal (machines can do better!)

Automated Machine Learning

- Democratizes the use of machine learning methods (no longer need expert domain knowledge)
- Save human time, effort, money
- Better performance

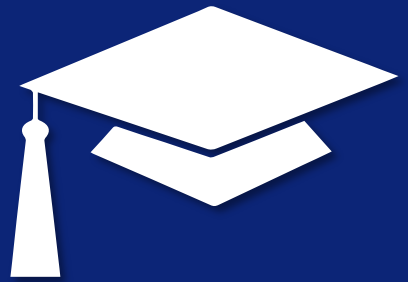


Automated Machine Learning



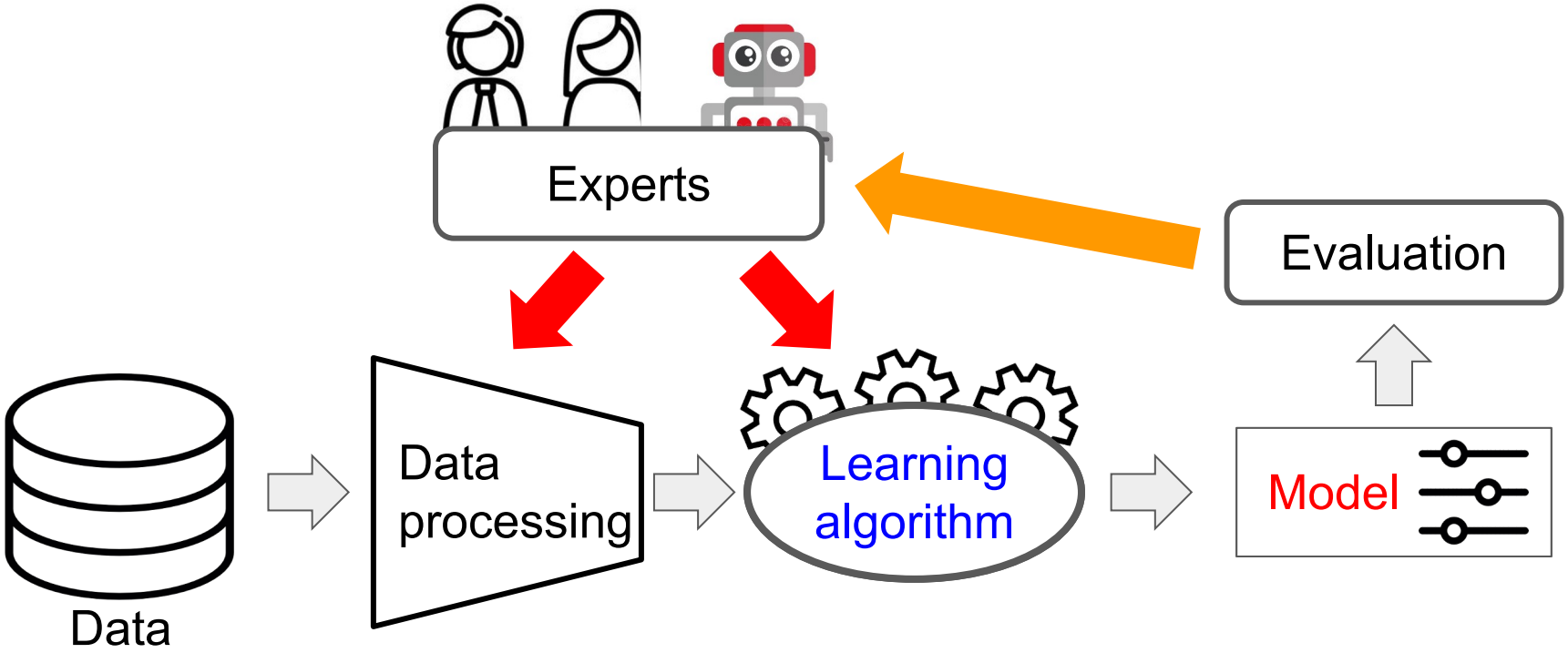
Universiteit Leiden

Any questions so far?

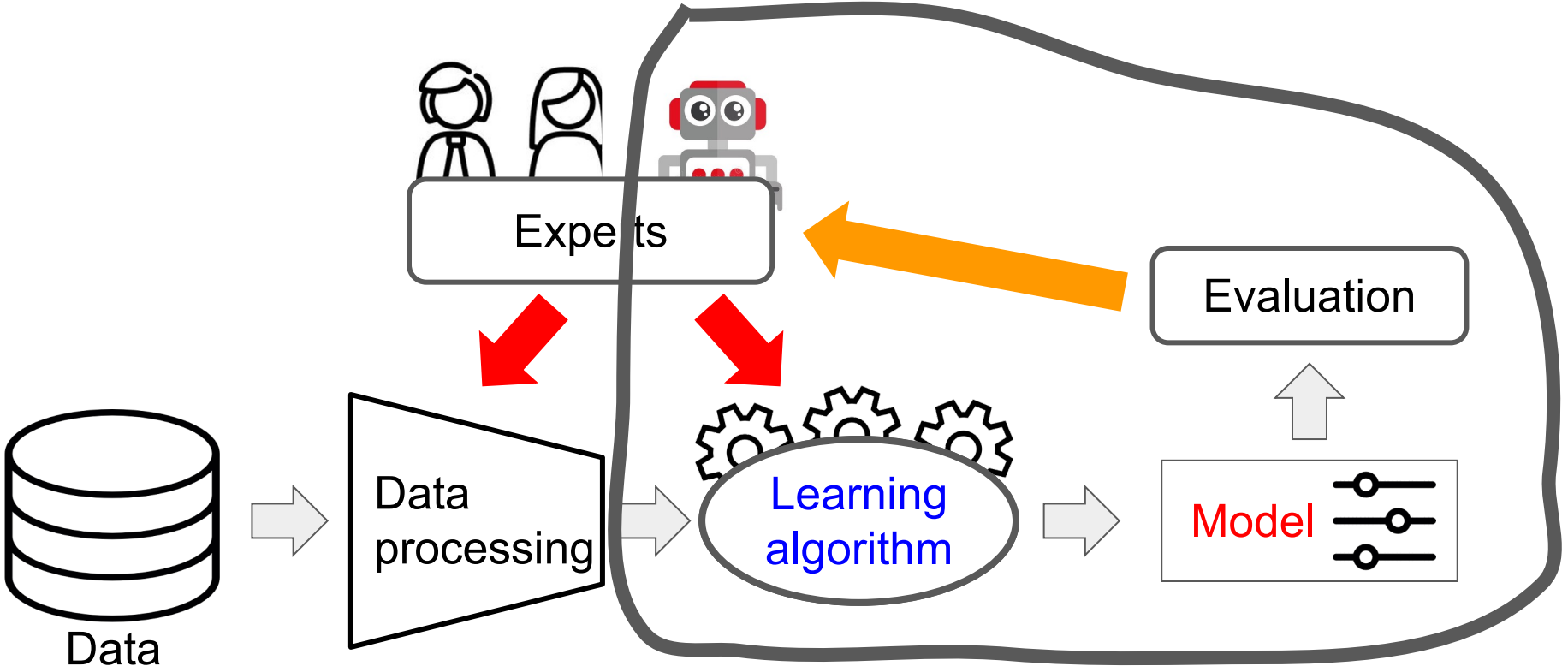


Combined Algorithm Selection and Hyperparameter optimization (CASH)

Combined Algorithm Selection and Hyperparameter optimization (CASH)



Combined Algorithm Selection and Hyperparameter optimization (CASH)



CASH - definition

Universiteit Leiden

CASH - definition

Given:

$\mathcal{T}$: task we want to solve,

CASH - definition

Given:

$\mathcal{T}$: task we want to solve, D_{train} : the training data

CASH - definition

Given:

$\mathcal{T}$: task we want to solve, D_{train} : the training data

$\mathcal{L}_{\mathcal{T}}$: the loss function (-1 * objective function)

CASH - definition

Given:

$\mathcal{T}$: task we want to solve, D_{train} : the training data

$\mathcal{L}_{\mathcal{T}}$: the loss function (-1 * objective function)

$\mathcal{A}$: the space of algorithms over which we search

CASH - definition

Given:

$\mathcal{T}$: task we want to solve, D_{train} : the training data

$\mathcal{L}_{\mathcal{T}}$: the loss function (-1 * objective function)

$\mathcal{A}$: the space of algorithms over which we search

Θ : the space of hyperparameters

CASH - definition

Given:

$\mathcal{T}$: task we want to solve, D_{train} : the training data

$\mathcal{L}_{\mathcal{T}}$: the loss function (-1 * objective function)

$\mathcal{A}$: the space of algorithms over which we search

Θ : the space of hyperparameters

Find: $$A^*, \theta^* = \underset{A \in \mathcal{A}, \theta \in \Theta}{arg\,min}\, \mathbb{E}_{p_{\mathcal{T}}} \left[\mathcal{L}_{\mathcal{T}}(A_\theta(D_{train})) \right]$$

CASH - definition

Given:

$\mathcal{T}$: task we want to solve, D_{train} : the training data

$\mathcal{L}_{\mathcal{T}}$: the loss function (-1 * objective function)

$\mathcal{A}$: the space of algorithms over which we search

Θ : the space of hyperparameters

Find: $$A^*, \theta^* = \underset{A \in \mathcal{A}, \theta \in \Theta}{arg\,min}\, \mathbb{E}_{p_{\mathcal{T}}} \left[\mathcal{L}_{\mathcal{T}} \left(A_\theta \left(D_{train} \right) \right) \right]$$

Output of learning algorithm = model

CASH - definition

Given:

$\mathcal{T}$: task we want to solve, D_{train} : the training data

$\mathcal{L}_{\mathcal{T}}$: the loss function (-1 * objective function)

$\mathcal{A}$: the space of algorithms over which we search

Θ : the space of hyperparameters

Find: $$A^*, \theta^* = \underset{A \in \mathcal{A}, \theta \in \Theta}{arg\,min}\, \mathbb{E}_{p_{\mathcal{T}}} \left[\mathcal{L}_{\mathcal{T}}(A_\theta(D_{train})) \right]$$

Loss of the model

CASH - definition

Given:

$\mathcal{T}$: task we want to solve, D_{train} : the training data

$\mathcal{L}_{\mathcal{T}}$: the loss function (-1 * objective function)

$\mathcal{A}$: the space of algorithms over which we search

Θ : the space of hyperparameters

Find: $$A^*, \theta^* = \underset{A \in \mathcal{A}, \theta \in \Theta}{arg\,min} \mathbb{E}_{p_{\mathcal{T}}} \left[\mathcal{L}_{\mathcal{T}}(A_\theta(D_{train})) \right]$$

Expected/Mean loss of the model

CASH - definition

Given:

$\mathcal{T}$: task we want to solve, D_{train} : the training data

$\mathcal{L}_{\mathcal{T}}$: the loss function (-1 * objective function)

$\mathcal{A}$: the space of algorithms over which we search

Θ : the space of hyperparameters

Find: $$A^*, \theta^* = \underset{A \in \mathcal{A}, \theta \in \Theta}{arg\,min}\, \mathbb{E}_{p_{\mathcal{T}}} \left[\mathcal{L}_{\mathcal{T}}(A_\theta(D_{train})) \right]$$

Hyperparameter vector $\theta = [\theta_1\ \theta_2\ \ldots\ \theta_n]$

E.g., [learning rate, momentum, use_batch_norm]

CASH - definition

Given:

$\mathcal{T}$: task we want to solve, D_{train} : the training data

$\mathcal{L}_{\mathcal{T}}$: the loss function (-1 * objective function)

$\mathcal{A}$: the space of algorithms over which we search

Θ : the space of hyperparameters

Find: $$A^*, \theta^* = \underset{A \in \mathcal{A}, \theta \in \Theta}{arg\,min}\, \mathbb{E}_{p_{\mathcal{T}}} \left[\mathcal{L}_{\mathcal{T}}(A_\theta(D_{train})) \right]$$

= Combined Algorithm and Hyperparameter optimization (CASH)

HPO vs CASH

Combined Algorithm Selection and Hyperparameter optimization (CASH)

$$A^*, \theta^* = \underset{A \in \mathcal{A}, \theta \in \Theta}{arg\,min}\, \mathbb{E}_{p_{\mathcal{T}}} \left[\mathcal{L}_{\mathcal{T}}(A_\theta(D_{train})) \right]$$

HPO vs CASH

Universiteit Leiden

Combined Algorithm Selection and Hyperparameter optimization (CASH)

$$A^*, \theta^* = \underset{A \in \mathcal{A},\, \theta \in \Theta}{arg\,min}\, \mathbb{E}_{p_{\mathcal{T}}} \left[\mathcal{L}_{\mathcal{T}}(A_\theta(D_{train})) \right]$$

Hyperparameter optimization (HPO)

$$\theta^* = \underset{\theta \in \Theta}{arg\,min}\, \mathbb{E}_{p_{\mathcal{T}}} \left[\mathcal{L}_{\mathcal{T}}(A_\theta(D_{train})) \right]$$

HPO vs CASH

Universiteit Leiden

Combined Algorithm Selection and Hyperparameter optimization (CASH)

$$A^*, \theta^* = \underset{A \in \mathcal{A}, \theta \in \Theta}{arg\,min}\, \mathbb{E}_{p_{\mathcal{T}}} \left[\mathcal{L}_{\mathcal{T}}(A_\theta(D_{train})) \right]$$

Hyperparameter optimization (HPO)

$$\theta^* = \underset{\theta \in \Theta}{arg\,min}\, \mathbb{E}_{p_{\mathcal{T}}} \left[\mathcal{L}_{\mathcal{T}}(A_\theta(D_{train})) \right]$$

HPO = CASH when including the algorithm as “hyperparameter”

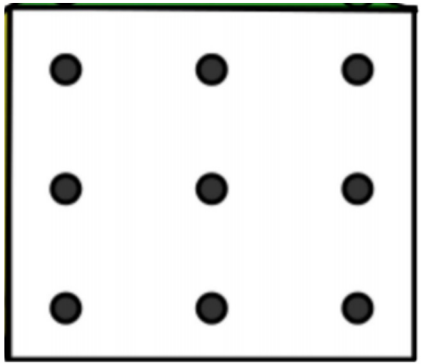
CASH and HPO methods

Methods we look at today:

- Grid search
- Random search
- Successive halving & Hyperband
- Bayesian Optimization (BO)

Grid search

Idea: define sets of values you want to try for some (or all) hyperparameters, evaluate all possible combinations

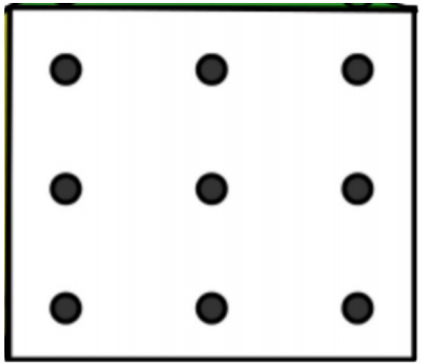


Grid search

Idea: define sets of values you want to try for some (or all) hyperparameters, evaluate all possible combinations

$$v(\theta_1) = \{0.1, 0.01, 0.001\},$$

Value sets:

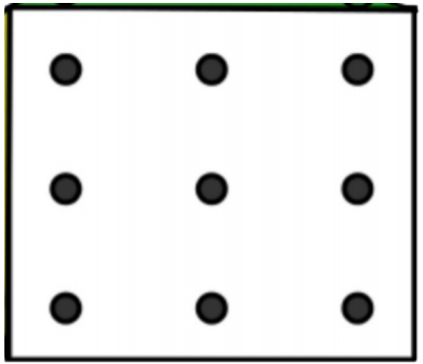


Grid search

Idea: define sets of values you want to try for some (or all) hyperparameters, evaluate all possible combinations

Value sets:

$$v(\theta_1) = \{0.1, 0.01, 0.001\},$$
$$v(\theta_2) = \{0.99, 0.9, 0.8\},$$

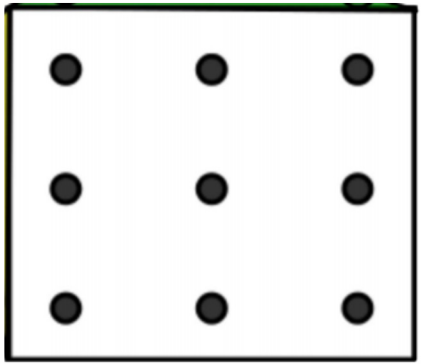


Grid search

Idea: define sets of values you want to try for some (or all) hyperparameters, evaluate all possible combinations

Value sets:

$$v(\theta_1) = \{0.1, 0.01, 0.001\},$$
$$v(\theta_2) = \{0.99, 0.9, 0.8\},$$
$$\dots$$
$$v(\theta_n) = \{True, False\}$$



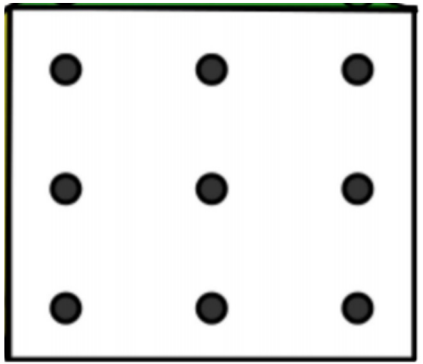
Grid search

Idea: define sets of values you want to try for some (or all) hyperparameters, evaluate all possible combinations

Value sets:

$$v(\theta_1) = \{0.1, 0.01, 0.001\},$$
$$v(\theta_2) = \{0.99, 0.9, 0.8\},$$
$$\dots$$
$$v(\theta_n) = \{True, False\}$$

Evaluate: $v(\theta_1) \times v(\theta_2) \times \dots \times v(\theta_n)$



Grid search

Pros:

- Simple
- Easy to parallelize

Cons:

- Waste of compute when one or more hyperparameters not important
- Scalability (exponential growth number of combinations)
- How to define the grid?

Grid search

Pros:

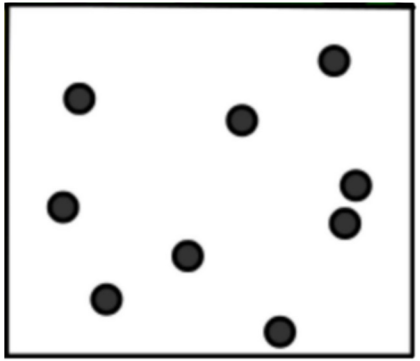
- Simple
- Easy to parallelize

Cons:

- Waste of compute when one or more hyperparameters not important
- Scalability (exponential growth number of combinations)
- How to define the grid?

Random search

Idea: define probability distributions on ranges of values for every hyperparameter, draw hyperparameter configurations and evaluate

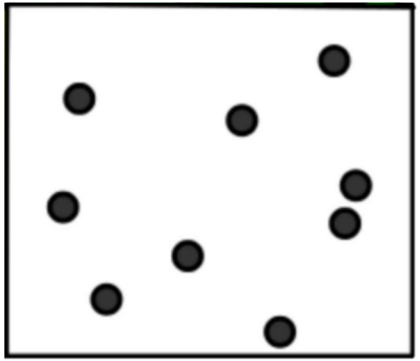


Random search

Idea: define probability distributions on ranges of values for every hyperparameter, draw hyperparameter configurations and evaluate

$$p(\theta_1) = LogUniform(0.001, 0.1)$$

Distributions:

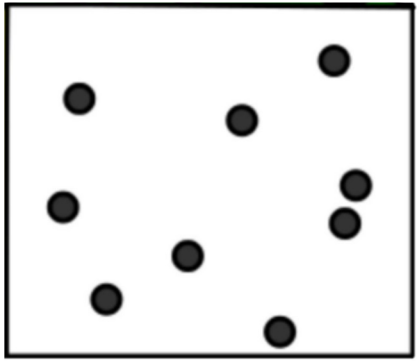


Random search

Idea: define probability distributions on ranges of values for every hyperparameter, draw hyperparameter configurations and evaluate

$$p(\theta_1) = LogUniform(0.001, 0.1)$$

Distributions: $p(\theta_2) = Uniform(0.8, 0.99)$



Random search

Idea: define probability distributions on ranges of values for every hyperparameter, draw hyperparameter configurations and evaluate

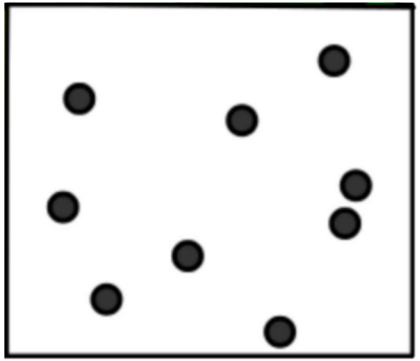
Distributions:

$$p(\theta_1) = LogUniform(0.001, 0.1)$$

$$p(\theta_2) = Uniform(0.8, 0.99)$$

$$\dots$$

$$p(\theta_n) = Categorical(\{True, False\})$$



Random search

Idea: define probability distributions on ranges of values for every hyperparameter, draw hyperparameter configurations and evaluate

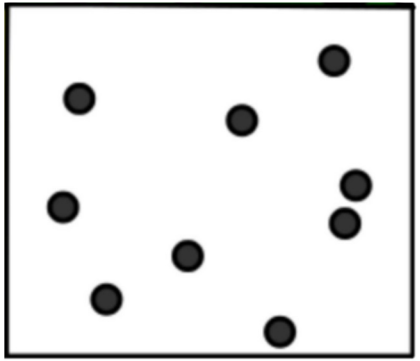
Distributions:

$$p(\theta_1) = LogUniform(0.001, 0.1)$$

$$p(\theta_2) = Uniform(0.8, 0.99)$$

$$\dots$$

$$p(\theta_n) = Categorical(\{True, False\})$$



Evaluate M configurations:

$$\theta \backsim p(\theta_1, \theta_2, \dots, \theta_n)$$

Random search

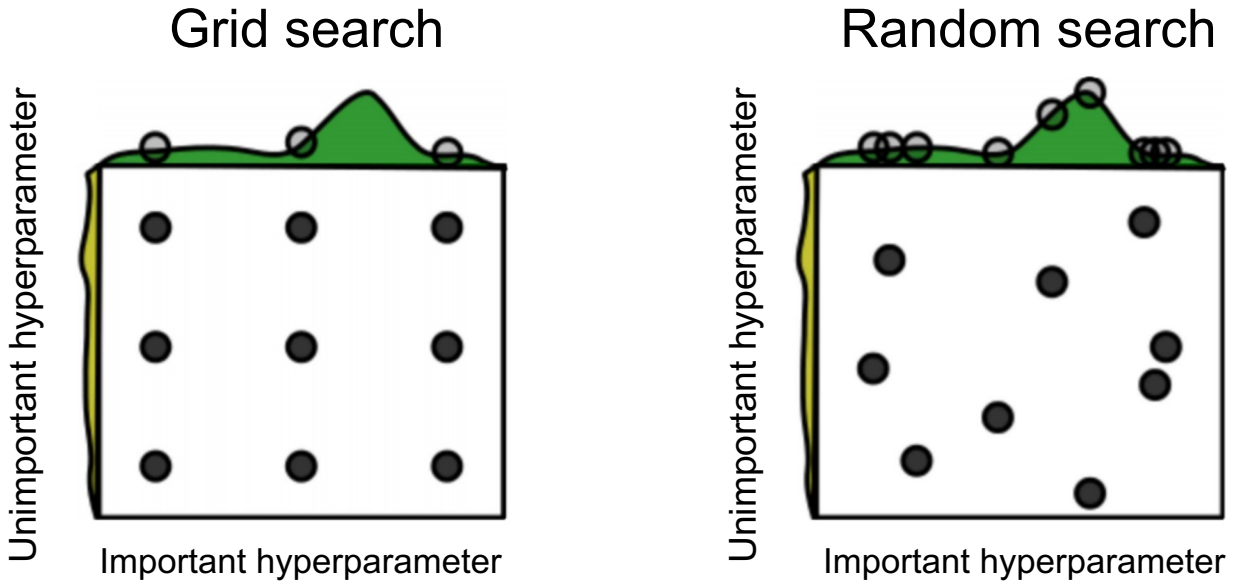
Pros:

- Simple
- Easy to parallelize
- Can set a budget of function evaluations (we evaluate M hyperparameter configurations)
- Theoretical guarantees of convergence with proper ranges

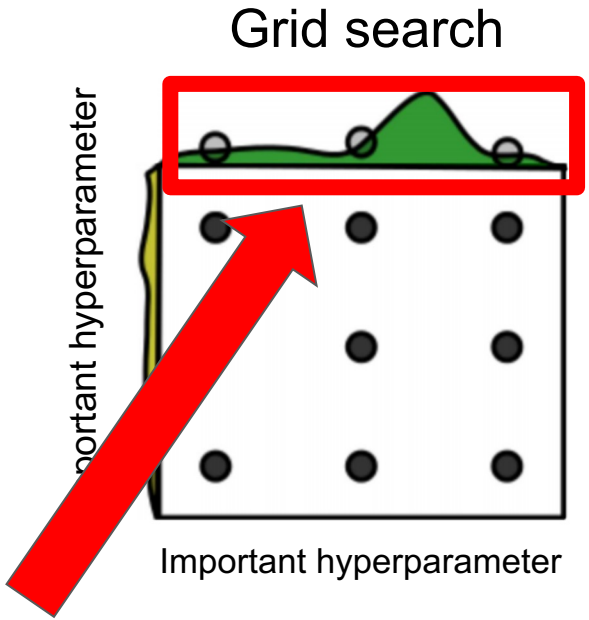
Cons:

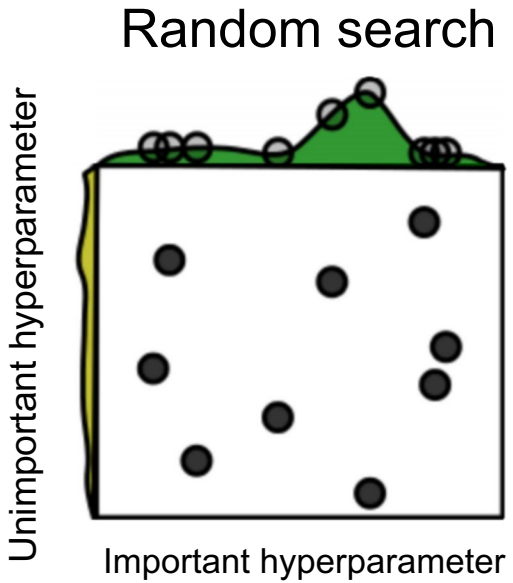
- Not data efficient
- Still expensive to perform

Grid search vs Random search



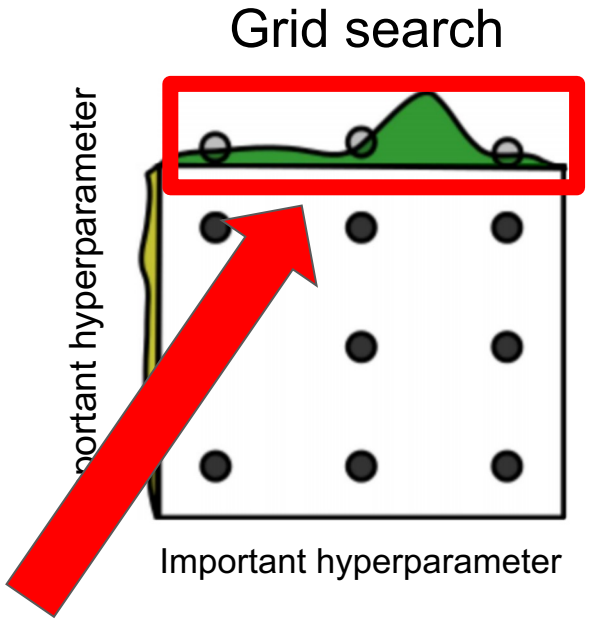
Grid search vs Random search



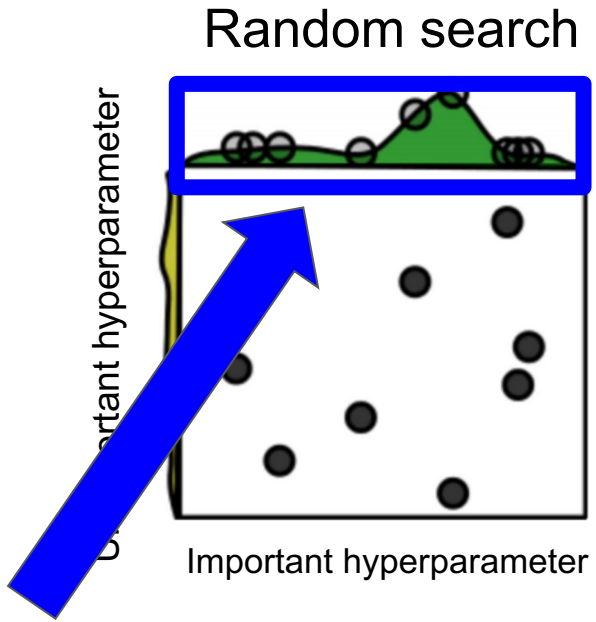


Only 3 useful evaluations

Grid search vs Random search

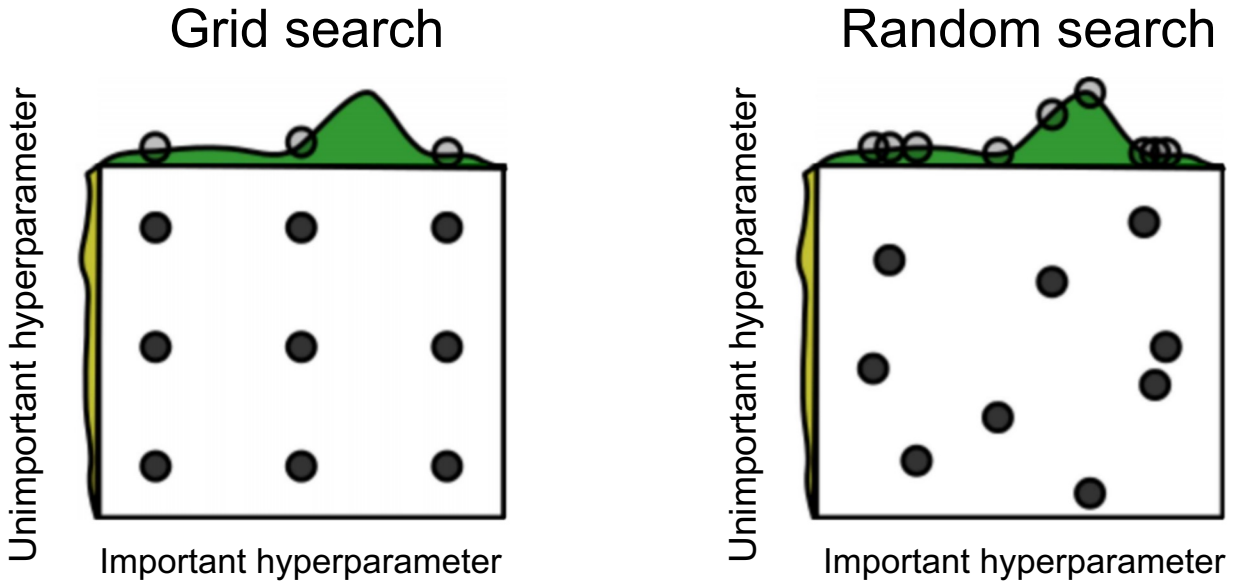


Only 3 useful evaluations



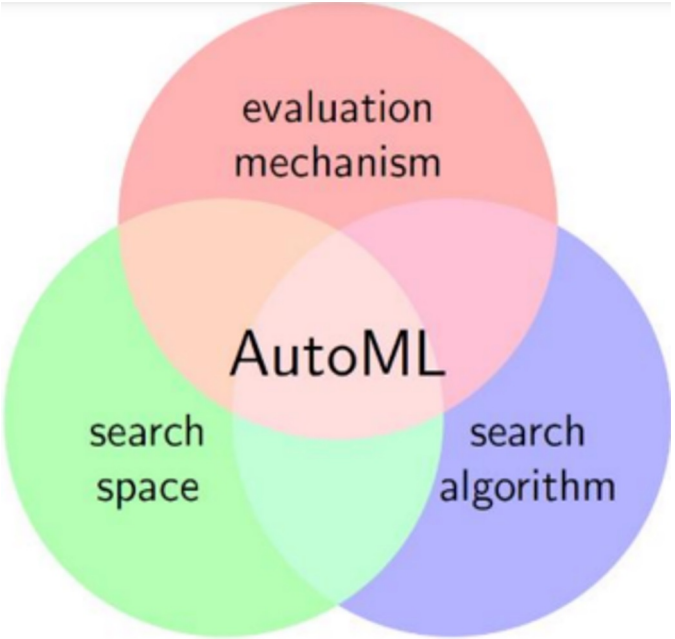
9 useful evaluations

Grid search vs Random search



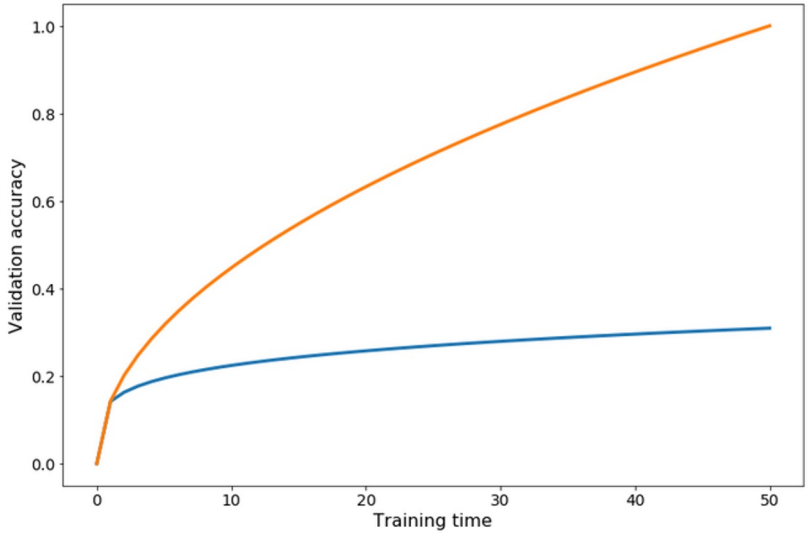
⇒ Random search is more efficient when there are unimportant hyperparameters (better **coverage**)

Automated Machine Learning



Grid search and random search

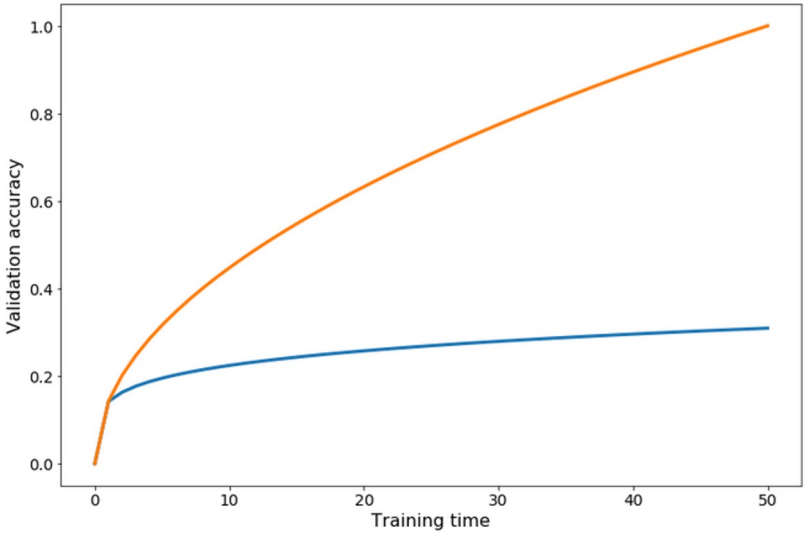
Downside: Every hyperparameter configuration is fully trained (expensive!)



Grid search and random search

Downside: Every hyperparameter configuration is fully trained (expensive!)

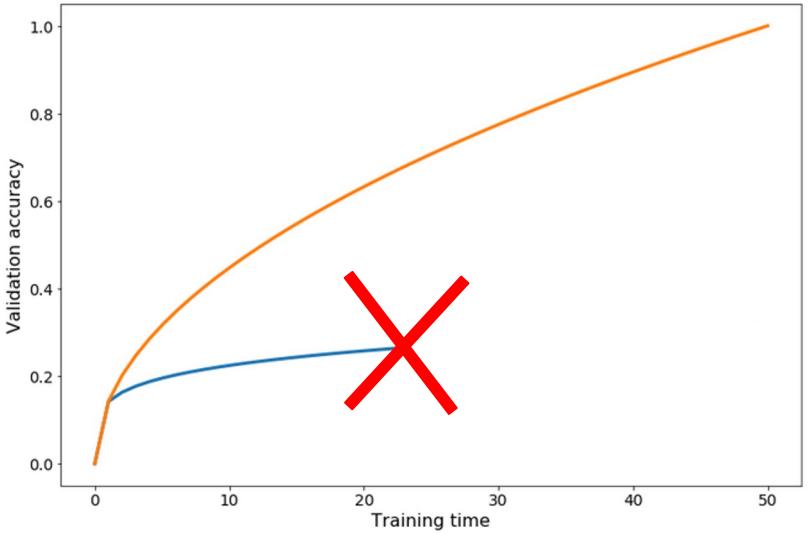
⇒ But maybe, we can **detect early** on that a configuration will be bad and **discard** it



Grid search and random search

Downside: Every hyperparameter configuration is fully trained (expensive!)

⇒ But maybe, we can **detect early** on that a configuration will be bad and **discard** it

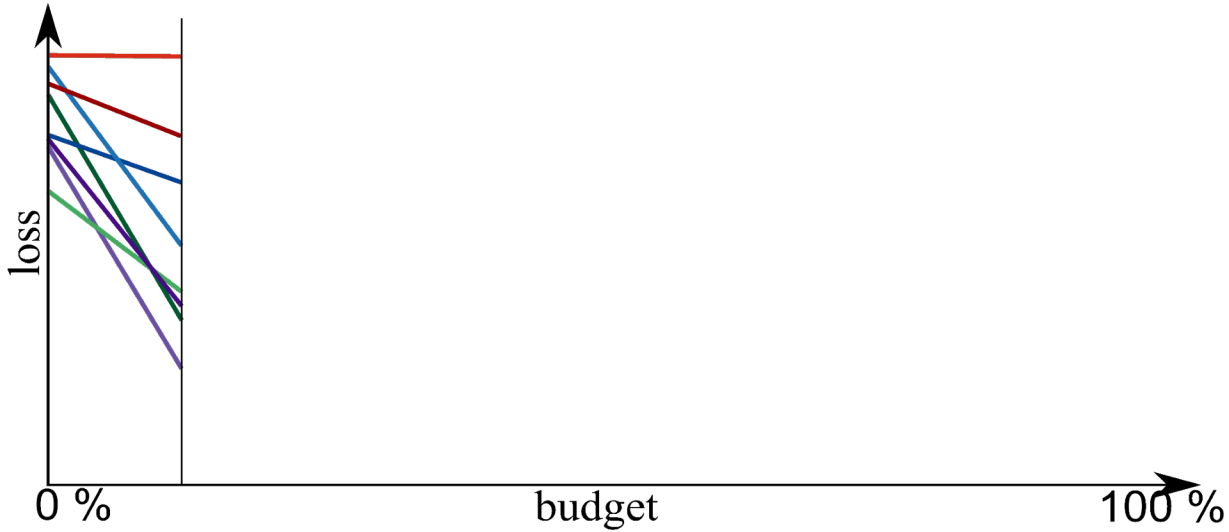


Techniques that do this

- Successive halving (Jamieson and Talwalkar, 2016)
- Hyperband (Li et al., 2018)

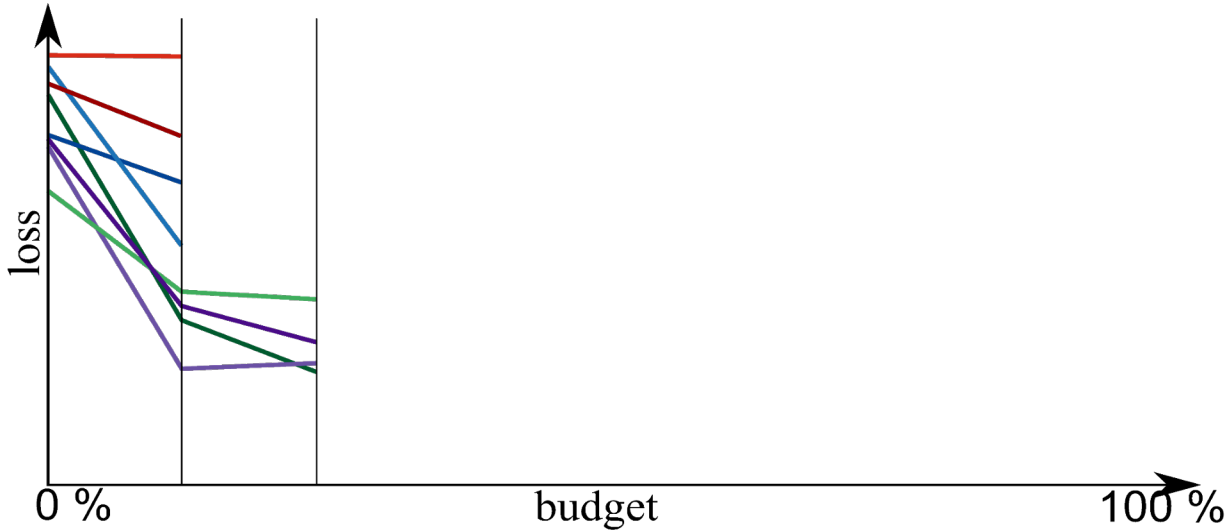
Successive halving

Successive halving



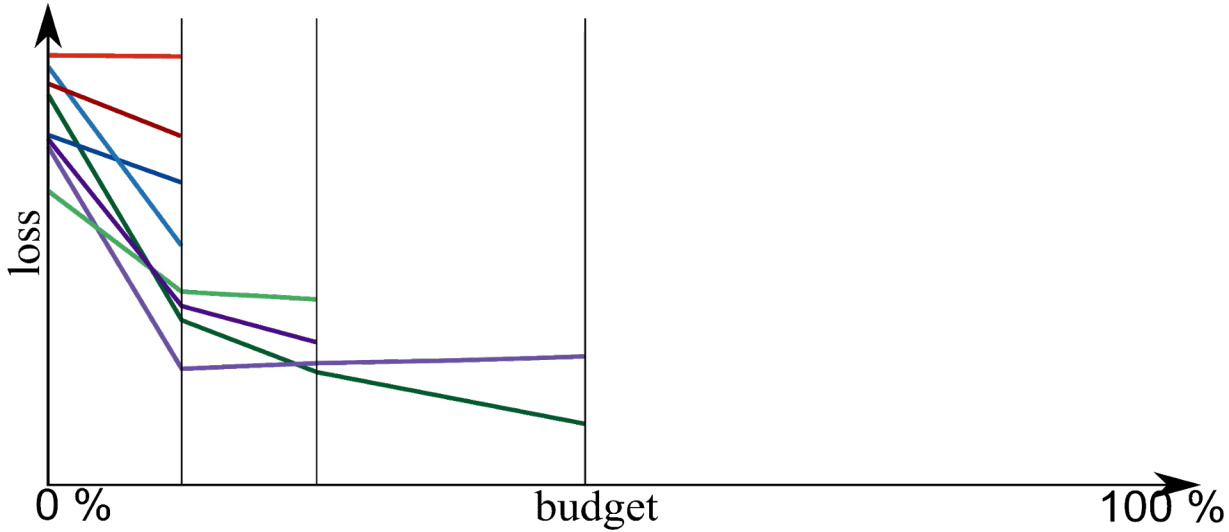
Round 0

Successive halving



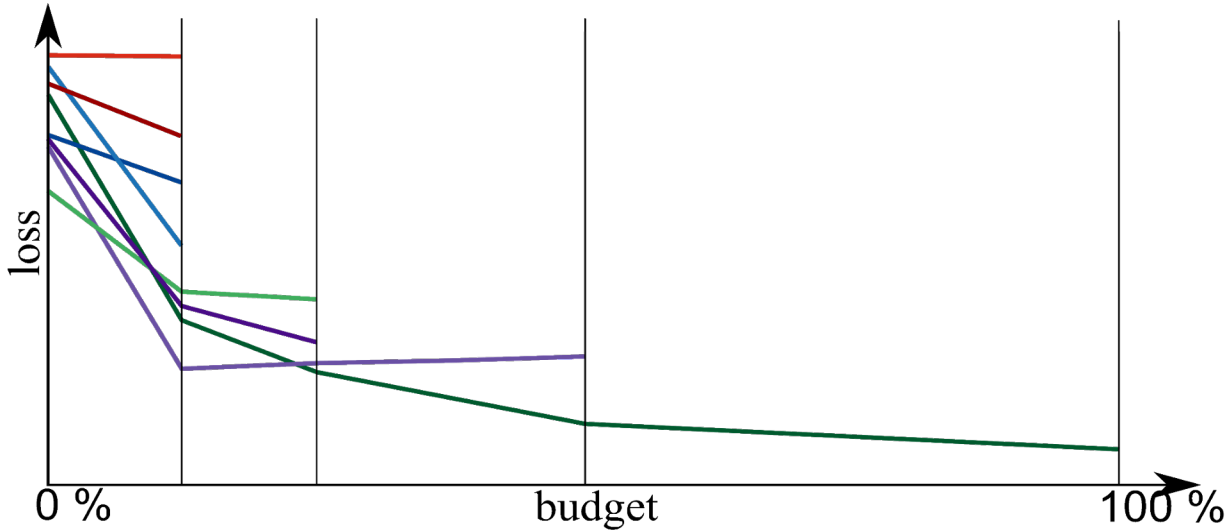
Round 1

Successive halving



Round 2

Successive halving



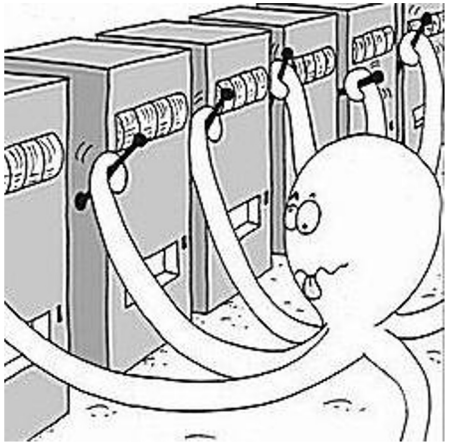
Round 3

Successive halving

This is a bandit-based method

- Different configurations are different one-armed slot machines
- Pull the lever = assign more budget and observe reward

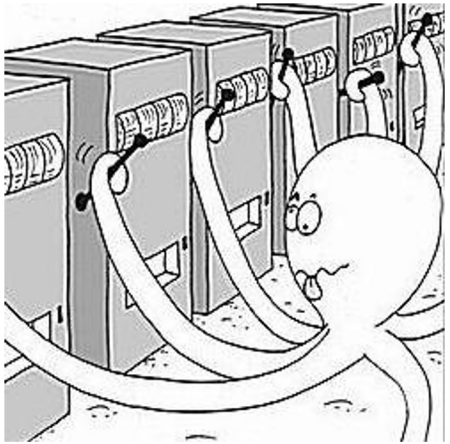
Goal: find the best slot machine (configuration)



Successive halving

Key idea:

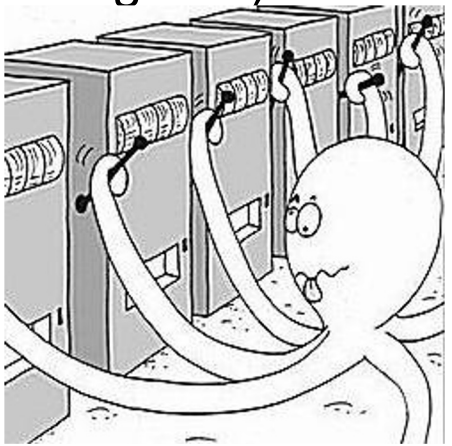
- Start with a set of n random initial configurations $\{A_\theta^{(i)}\}_{i=1}^n$ and a budget B



Successive halving

Key idea:

- Start with a set of n random initial configurations $\{A_\theta^{(i)}\}_{i=1}^{n}$ and a budget B
- Every round the candidate pool size is divided by γ (halving rate)

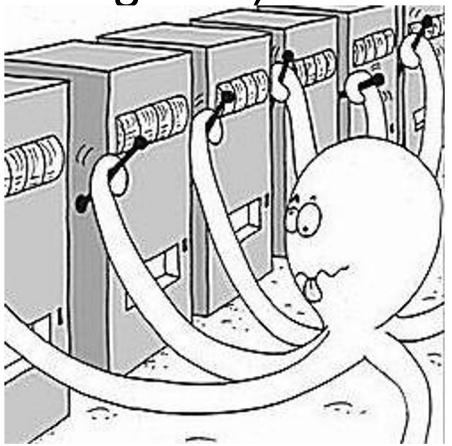


Successive halving

Key idea:

- Start with a set of n random initial configurations $\{A_\theta^{(i)}\}_{i=1}^{n}$ and a budget B
- Every round the candidate pool size is divided by γ (halving rate)

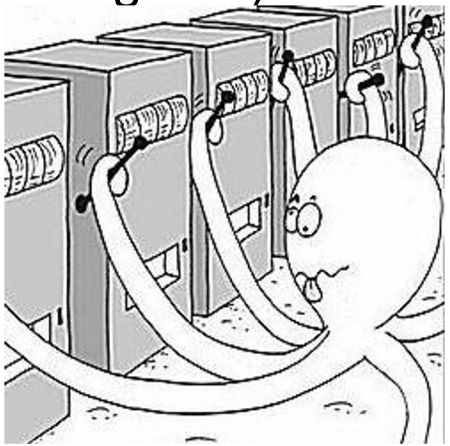
How many rounds are there if we start with n configurations?



Successive halving

Key idea:

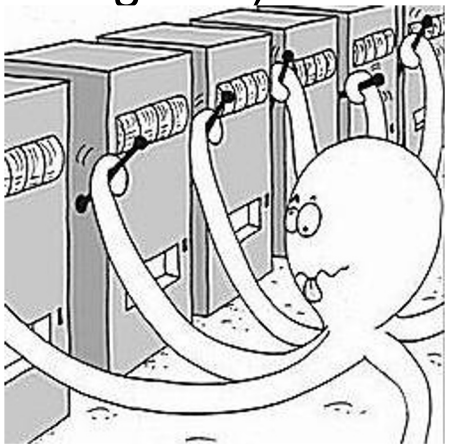
- Start with a set of n random initial configurations $\{A_\theta^{(i)}\}_{i=1}^{n}$ and a budget B
- Every round the candidate pool size is divided by γ (halving rate)
- Perform $\log_\gamma(n)$ rounds



Successive halving

Key idea:

- Start with a set of n random initial configurations $\{A_\theta^{(i)}\}_{i=1}^{n}$ and a budget B
- Every round the candidate pool size is divided by γ (halving rate)
- Perform $\log_\gamma(n)$ rounds
- Budget is divided uniformly over rounds

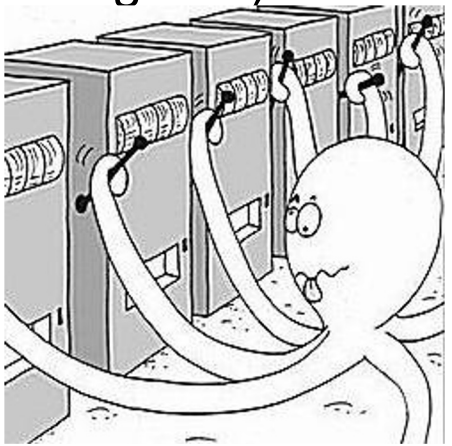


Successive halving

Key idea:

- Start with a set of n random initial configurations $\{A_\theta^{(i)}\}_{i=1}^{n}$ and a budget B
- Every round the candidate pool size is divided by γ (halving rate)
- Perform $\log_\gamma(n)$ rounds
- Budget is divided uniformly over rounds

⇒ We assign exponentially more budget to good configs



Successive halving

Number of initial configs, n=64. Total budget B=384

Halving rate $\gamma = 2$

Successive halving

Number of initial configs, n=64. Total budget B=384

Halving rate γ = 2

k	$\lvert S_k \rvert$	r_k
0		
1		
2		
3		
4		
5		

$\lvert S_k \rvert$: size of candidate pool in round k

r_k : budget in round k per configuration

Successive halving

Number of initial configs, n=64. Total budget B=384

Halving rate $\gamma = 2$

k	$\lvert S_k \rvert$	r_k
0	64	1
1	32	2
2	16	4
3	8	8
4	4	16
5	2	32

Successive halving

Effects of the budget and halving rate

- Higher budget ⇒ Higher budget per config (*more informed decisions*)

Successive halving

Effects of the budget and halving rate

- Higher budget ⇒ Higher budget per config (*more informed decisions*)
- Higher halving rate ⇒ More aggressive pruning (*could be too aggressive and disregard the best candidate early on)*

Successive halving

Pros

- Allows us to work with fixed budget
- Good empirical results
- Strong theoretical foundations
- Parallelizable

Cons

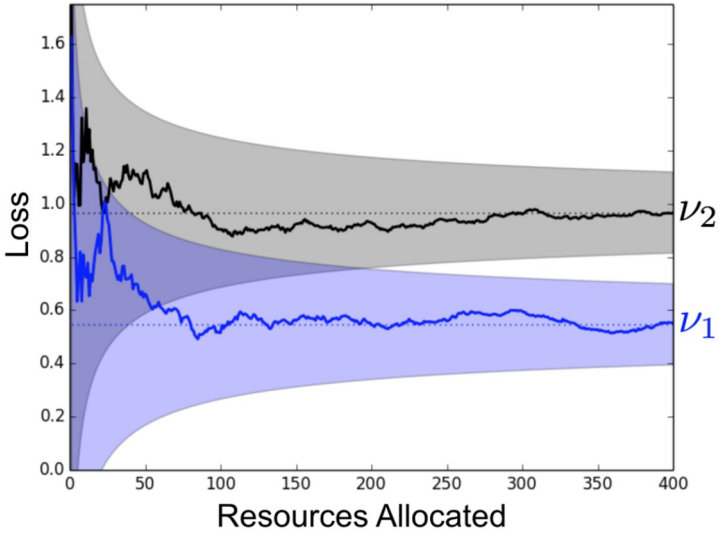
- Learning curves can cross (good candidates to be discarded early)
- n/B trade-off

Successive halving

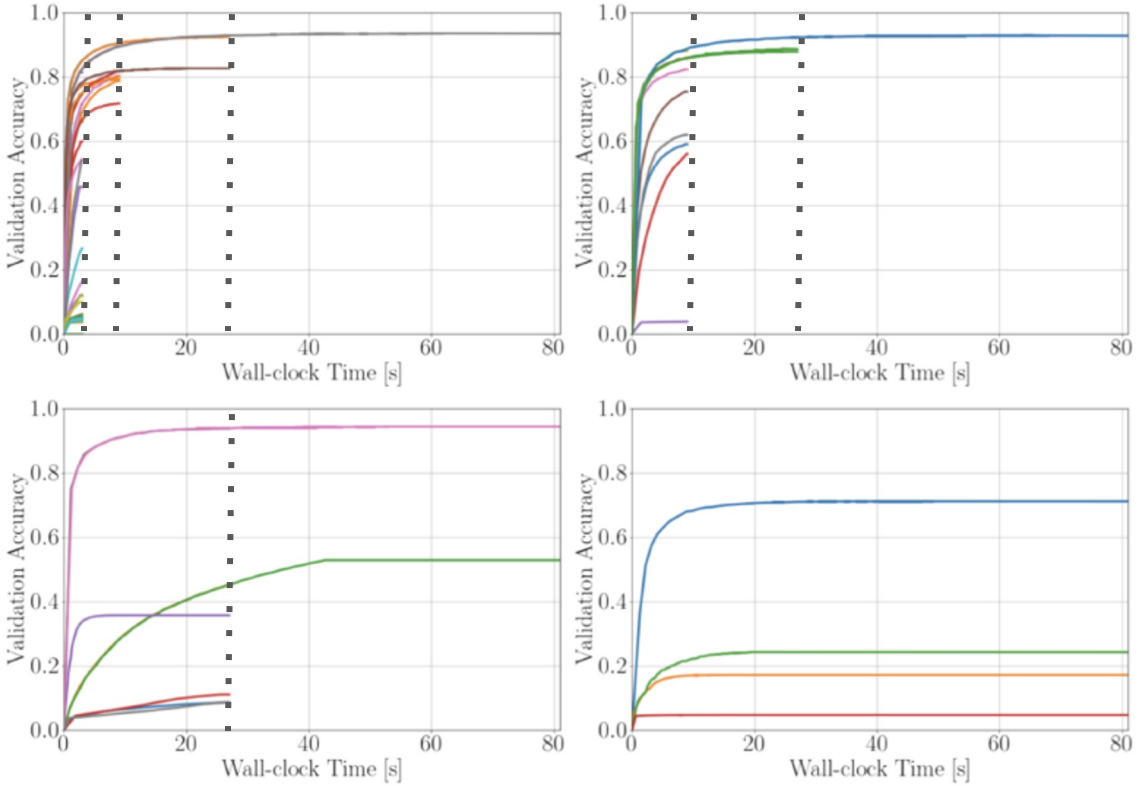
Trade-off: Given a fixed budget B

- Large *n* (num initial configs)? ⇒ small amount of time per configuration
- Small *n?* ⇒ larger amount of time per configuration

⇒ We initially do not know what works best



Hyperband


Validation Accuracy
Wall-clock Time [s]
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80

Hyperband

Idea: Perform different successive halving "*brackets*" $s_{max} + 1$ times with various initial candidate pool sizes *n* for a fixed budget B

Hyperband

Idea: Perform different successive halving "*brackets*" $s_{max} + 1$ times with various initial candidate pool sizes *n* for a fixed budget B

- Start with large *n* (exploration, *less time per configuration*)

Hyperband

Idea: Perform different successive halving "*brackets*" $s_{max} + 1$ times with various initial candidate pool sizes *n* for a fixed budget B

- Start with large *n* (exploration, *less time per configuration*)
- Every consecutive bracket: reduce *n (more time per configuration)*

Hyperband

Idea: Perform different successive halving "*brackets*" $s_{max} + 1$ times with various initial candidate pool sizes *n* for a fixed budget B

- Start with large *n* (exploration, *less time per configuration*)
- Every consecutive bracket: reduce *n (more time per configuration)*

Hyperparameters:

- R: maximum resource that can be allocated to a single technique within a round of successive halving

Hyperband

Idea: Perform different successive halving "*brackets*" $s_{max} + 1$ times with various initial candidate pool sizes *n* for a fixed budget B

- Start with large *n* (exploration, *less time per configuration*)
- Every consecutive bracket: reduce *n (more time per configuration)*

Hyperparameters:

- R: maximum resource that can be allocated to a single technique within a round of successive halving
- γ: controls proportion of discarded configs every round

Hyperband

i
0
1
2
3
4

$$\gamma = 3$$

Hyperband

	$s = 4$	
i	n_i	r_i
0	81	1
1	27	3
2	9	9
3	3	27
4	1	81

First bracket with n=81 (we sample random configurations). Just successive halving.

$$\gamma = 3$$

Hyperband

	$s=4$		$s=3$	
i	n_i	r_i	n_i	r_i
0	81	1	**34**	3
1	27	3	**11**	9
2	9	9	3	27
3	3	27	1	81
4	1	81		

Second bracket with n=34 (new randomly sampled configurations). Just successive halving.

$$\gamma = 3$$

Hyperband

	$s=4$		$s=3$		$s=2$	
i	n_i	r_i	n_i	r_i	n_i	r_i
0	81	1	**34**	3	**15**	9
1	27	3	**11**	9	**5**	27
2	9	9	3	27	1	81
3	3	27	1	81		
4	1	81				

$$\gamma = 3$$

Hyperband

	$s=4$		$s=3$		$s=2$		$s=1$	
i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i
0	81	1	**34**	3	**15**	9	**8**	27
1	27	3	**11**	9	**5**	27	2	81
2	9	9	3	27	1	81		
3	3	27	1	81				
4	1	81						

Various brackets, with $R = 81$ and $\gamma = 3$

Hyperband

	$s=4$		$s=3$		$s=2$		$s=1$		$s=0$	
i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i
0	81	1	**34**	3	**15**	9	**8**	27	5	81
1	27	3	**11**	9	**5**	27	2	81		
2	9	9	3	27	1	81				
3	3	27	1	81						
4	1	81								

Hyperband

	$s=4$		$s=3$		$s=2$		$s=1$		$s=0$	
i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i
0	81	1	**34**	3	**15**	9	**8**	27	5	81
1	27	3	**11**	9	**5**	27	2	81		
2	9	9	3	27	1	81				
3	3	27	1	81						
4	1	81								

No more intermediate discarding takes place
⇒ **this is random search!!!**

Hyperband

Overview of Hyperband

- Hyperband is a **logfactor** slower than random search to identify the best configuration (effect can be large!!)

Hyperband

Overview of Hyperband

- Hyperband is a **logfactor** slower than random search to identify the best configuration (effect can be large!!)
- Random search eventually converges to a solution, and so does Hyperband (the former is embedded in the latter)

Hyperband

Overview of Hyperband

- Hyperband is a **logfactor** slower than random search to identify the best configuration (effect can be large!!)
- Random search eventually converges to a solution, and so does Hyperband (the former is embedded in the latter)
- We explore different values of n for a fixed B ⇒ resolves b/N trade-off in successive halving

Limitation of techniques so far

They **randomly** select (initial) candidates to evaluate

Limitation of techniques so far

They **randomly** select (initial) candidates to evaluate

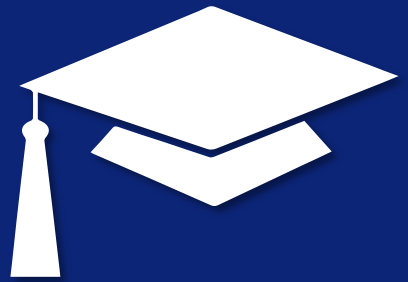
But if we have already evaluated a set of candidates, can’t we do better than random selection?

Limitation of techniques so far

They **randomly** select (initial) candidates to evaluate

But if we have already evaluated a set of candidates, can't we do better than random selection?

⇒ **Predict promising regions based on history of observations**



Sequential model-based optimization (SMBO)

Sequential model-based optimization

Key idea: Learn a surrogate model that for every configuration $\theta \in \Theta$:

- Predicts the loss (or performance)
- And provides a measure of uncertainty about the prediction

Sequential model-based optimization

Key idea: Learn a surrogate model that for every configuration $\theta \in \Theta$:

- Predicts the loss (or performance)
- And provides a measure of uncertainty about the prediction

$\Rightarrow$ Loosely speaking, the surrogate models $P(y|\theta)$

Sequential model-based optimization

⇒ Update the surrogate model sequentially as new observations (tried configuration, performance) arrive

Sequential model-based optimization

⇒ Update the surrogate model sequentially as new observations (tried configuration, performance) arrive

⇒ Use the surrogate model to select new promising configurations to evaluate

Sequential model-based optimization

⇒ Update the surrogate model sequentially as new observations (tried configuration, performance) arrive

⇒ Use the surrogate model to select new promising configurations to evaluate

Bayesian optimization:

- ❖ Start with a prior distribution

Sequential model-based optimization

⇒ Update the surrogate model sequentially as new observations (tried configuration, performance) arrive

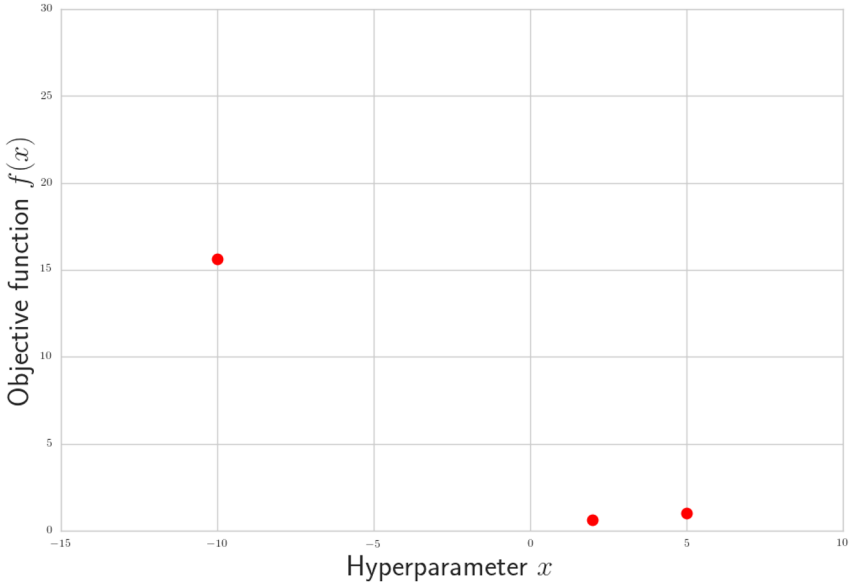
⇒ Use the surrogate model to select new promising configurations to evaluate

Bayesian optimization:

- ❖ Start with a prior distribution
- ❖ Repeat: Update our beliefs about the loss surface

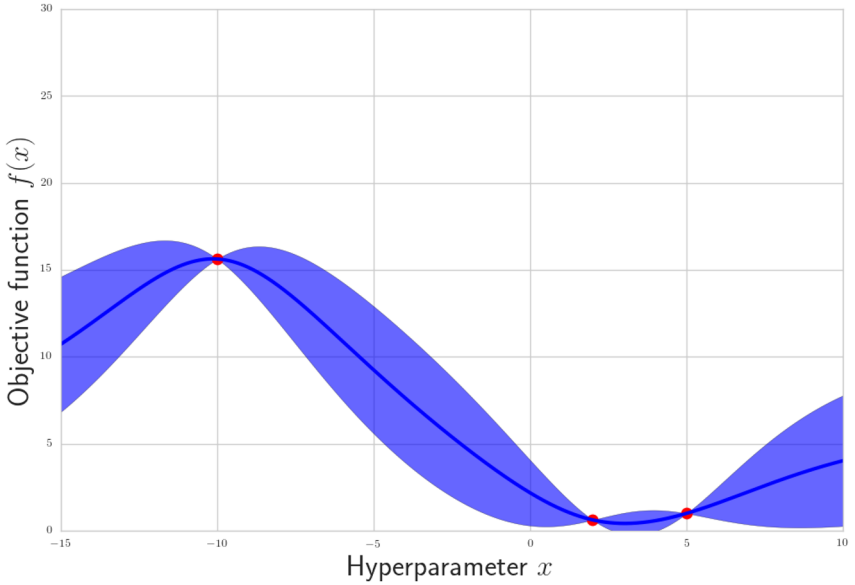
Visualization of Bayesian optimization

1. Fit a surrogate model to observations



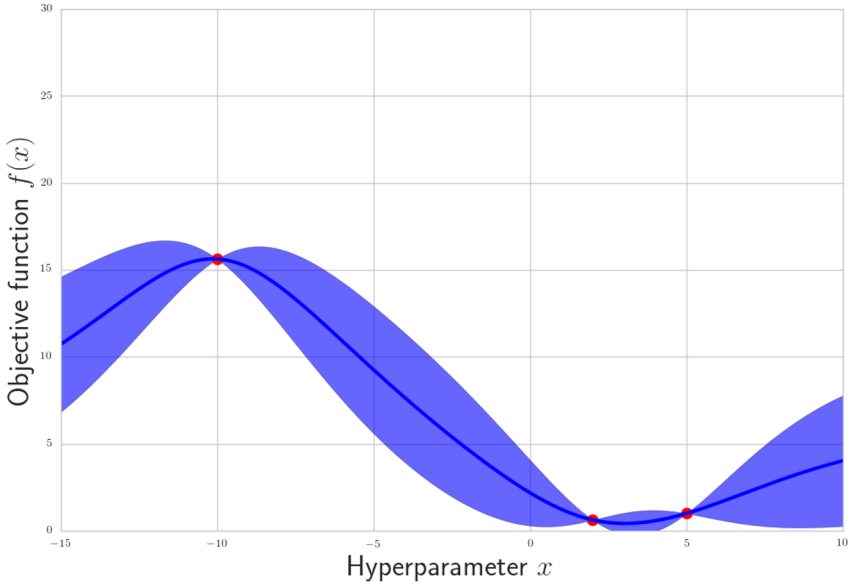
Visualization of Bayesian optimization

1. Fit a surrogate model to observations



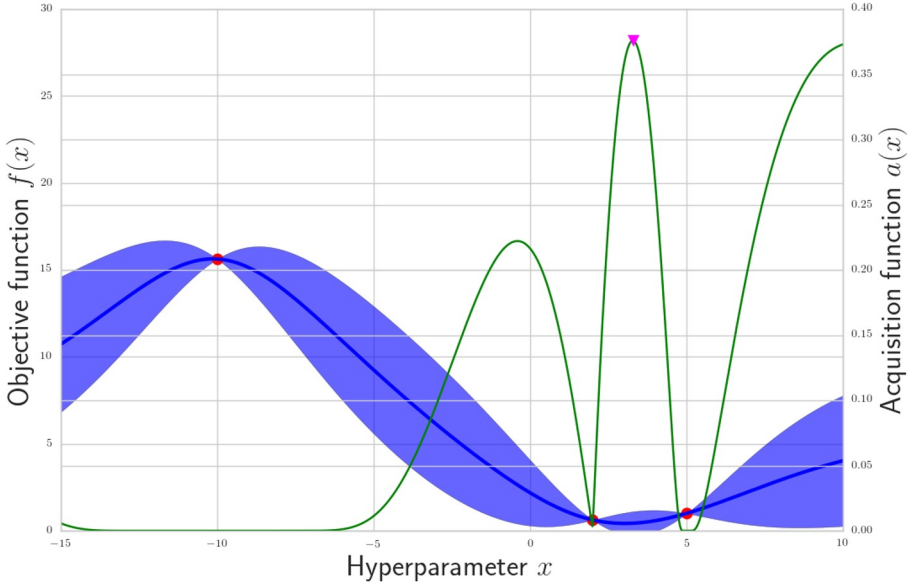
Visualization of Bayesian optimization

1. Fit a surrogate model to observations
2. Compute the **acquisition function** (estimated quality) and select new points accordingly



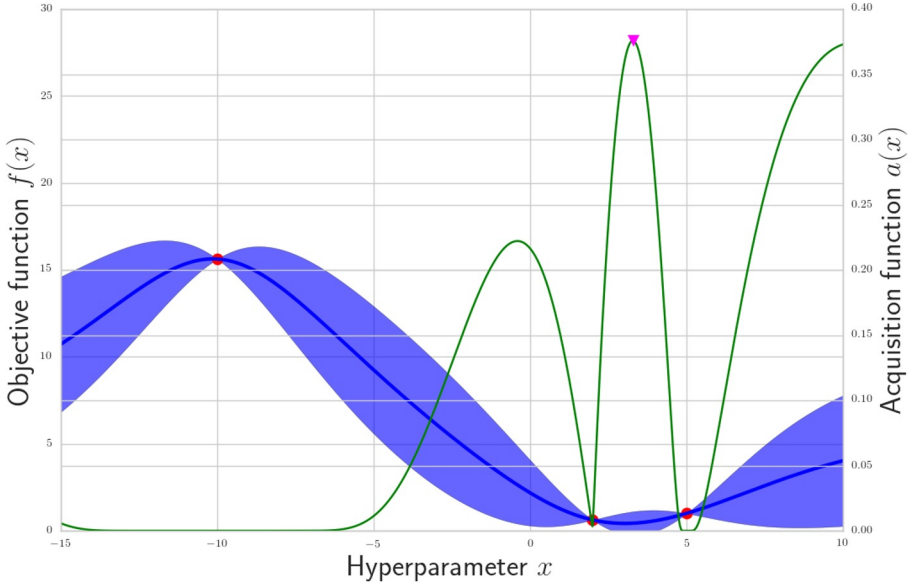
Visualization of Bayesian optimization

1. Fit a surrogate model to observations
2. Compute the **acquisition function** (estimated quality) and select new points accordingly



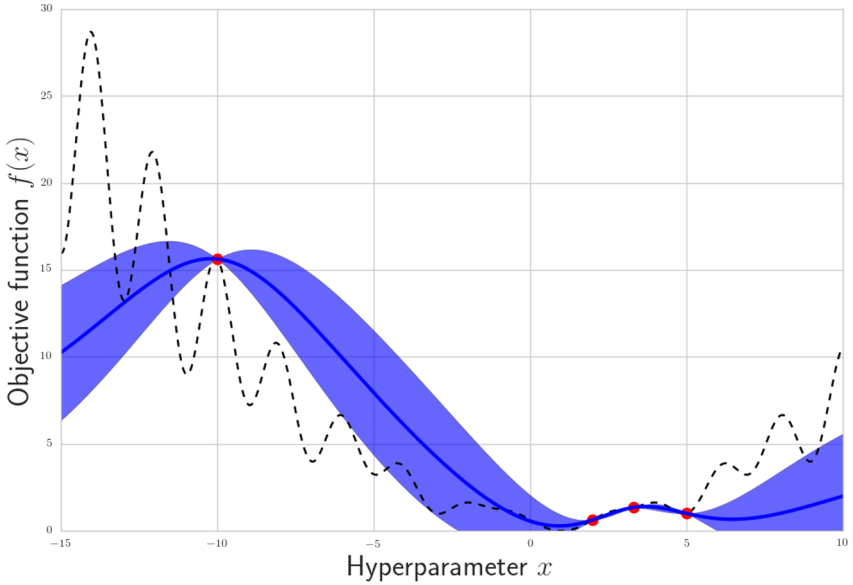
Visualization of Bayesian optimization

1. Fit a surrogate model to observations
2. Compute the **acquisition function** (estimated quality) and select new points accordingly
3. Evaluate new points



Visualization of Bayesian optimization

1. Fit a surrogate model to observations
2. Compute the **acquisition function** (estimated quality) and select new points accordingly
3. Evaluate new points
4. Repeat



Surrogate model - Bayesian optimization

What to use as surrogate model?

- Random forest (categorical and conditional hyperparameters)
- Gaussian process (good confidence bounds - only numerical hyperparameters)

Acquisition function - Bayesian optimization

Many different choices:

- Probability improvement
- Entropy search
- Lower/upper confidence bounds, UCB
- Expected Improvement (EI)

Goal: Compute how promising candidate configurations are

Bayesian optimization

Pros

- Data efficient (surrogate model)
- Convincingly faster than random search

Con: Training of surrogate model (extra time + how to set hyperparameters?)

Extension: Combine with meta-learning (warm-starting the surrogate model)

Overview

Grid Search

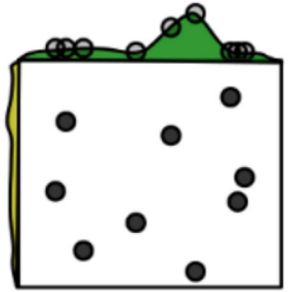
\+

Very simple approach
Can be used to study the problem

\-

Does not scale to high dimensions
Grid needs to be defined

Random Search



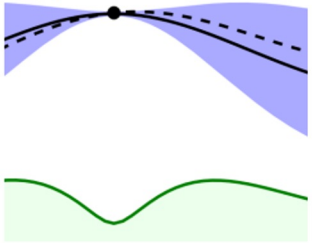
\+

Even more simple
Easily parallelizable
Eventually converges to optimum

\-

Not data efficient
Computationally expensive

Bayesian Optimization



\+

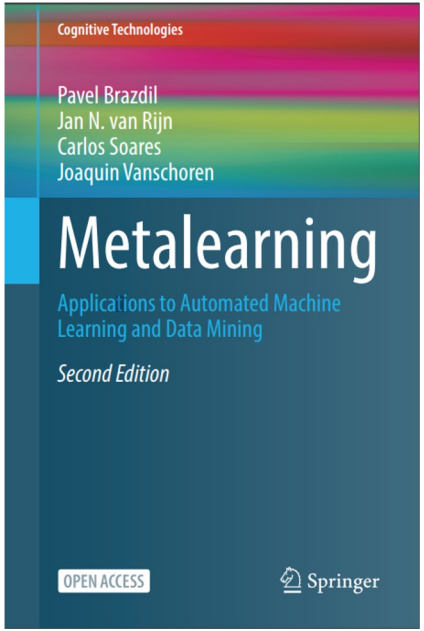
Black-box optimization
Data efficient
State of the art

\-

Not easy to parallelize

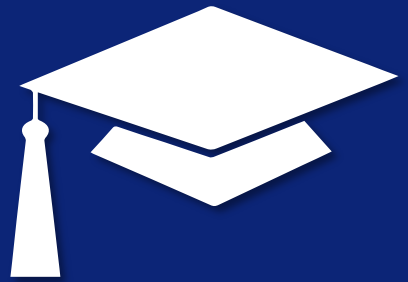
Relevant literature

- Metalearning book (Brazdil et al., 2022)
- Algorithms for Hyper-Parameter Optimization (Bergstra et al., 2011)
- Successive halving paper (Jamieson & Talwalkar, 2016)
- Hyperband paper (Li et al., 2016)
- Blog post on Hyperband



Acknowledgements

- Image by Juergen Schmidhuber
- Animation of TPE made by Alois Bissuel



Appendix

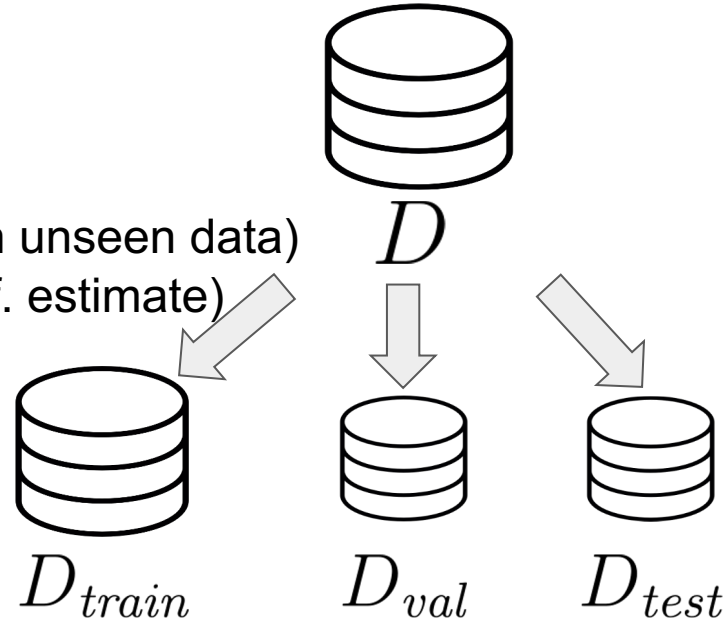
Train/val/test splits

How to compute $\mathbb{E}_{p_{\mathcal{T}}}\left[\mathcal{L}_{\mathcal{T}}(A_\theta(D_{train}))\right]$?

⇒ **Split dataset into 3 parts**

- Training set (fed into learning algorithm)
- Validation set (estimate above quantity on unseen data)
- Test set (never touched, only for final perf. estimate)

⇒ **Goal:** minimize validation loss



Successive halving

Input : Budget B, n arms where $\ell_{i,k}$ denotes the k^{th} loss from the i^{th} *arm*

Successive halving

Input : Budget B, n arms where $\ell_{i,k}$ denotes the k^{th} loss from the i^{th} *arm*

1 Initialize: $S_0 = \{1, 2, \ldots, n\}$ # Initial hyperparameter configurations

Successive halving

Input : Budget B, n arms where $\ell_{i,k}$ denotes the k^{th} loss from the i^{th} *arm*

Initialize: $S_0 = \{1, 2, \ldots, n\}$

for $k = 0, 1, \ldots, \lceil \log_2(n) \rceil - 1$ **do**

  Pull each arm in S_k for $r_k = \lfloor \frac{B}{|S_k| \lceil \log_2(n) \rceil} \rfloor$ additional times and set $R_k = \sum_{j=0}^{k} r_j$

Successive halving

Input : Budget B, n arms where $\ell_{i,k}$ denotes the k^{th} loss from the i^{th} *arm*

Initialize: $S_0 = \{1, 2, \ldots, n\}$

for $k = 0, 1, \ldots, \lceil \log_2(n) \rceil - 1$ **do**

  Pull each arm in S_k for $r_k = \lfloor \frac{B}{|S_k| \lceil \log_2(n) \rceil} \rfloor$ additional times and set $R_k = \sum_{j=0}^{k} r_j$

  Let σ_k be a bijection on S_k such that $\ell_{\sigma_k(1),R_k} \leq \ell_{\sigma_k(2),R_k} \leq \ldots \leq \ell_{\sigma_k(|S_k|),R_k}$

Order configurations by loss

Successive halving

Input : Budget B, n arms where $\ell_{i,k}$ denotes the k^{th} loss from the i^{th} *arm*

Initialize: $S_0 = \{1, 2, \ldots, n\}$

for $k = 0, 1, \ldots, \lceil \log_2(n) \rceil - 1$ **do**

  Pull each arm in S_k for $r_k = \lfloor \frac{B}{|S_k| \lceil \log_2(n) \rceil} \rfloor$ additional times and set $R_k = \sum_{j=0}^{k} r_j$

  Let σ_k be a bijection on S_k such that $\ell_{\sigma_k(1),R_k} \leq \ell_{\sigma_k(2),R_k} \leq \ldots \leq \ell_{\sigma_k(|S_k|),R_k}$

  $S_{k+1} = \{i \in S_k : \ell_{\sigma_k(i),R_k} \leq \ell_{\sigma_k(\lfloor |S_k|/2 \rfloor),R_k}\}$ # Best half continues to next round

end

Successive halving

Input : Budget B, n arms where $\ell_{i,k}$ denotes the k^{th} loss from the i^{th} *arm*

Initialize: $S_0 = \{1, 2, \ldots, n\}$

for $k = 0, 1, \ldots, \lceil \log_2(n) \rceil - 1$ **do**

  Pull each arm in S_k for $r_k = \lfloor \frac{B}{|S_k| \lceil \log_2(n) \rceil} \rfloor$ additional times and set $R_k = \sum_{j=0}^{k} r_j$

  Let σ_k be a bijection on S_k such that $\ell_{\sigma_k(1),R_k} \leq \ell_{\sigma_k(2),R_k} \leq \ldots \leq \ell_{\sigma_k(|S_k|),R_k}$

  $S_{k+1} = \{i \in S_k : \ell_{\sigma_k(i),R_k} \leq \ell_{\sigma_k(\lfloor |S_k|/2 \rfloor),R_k}\}$

end

Output: Singleton element of $S_{\lceil \log_2(n) \rceil}$

Successive halving

Input : Budget B, n arms where $\ell_{i,k}$ denotes the k^{th} loss from the i^{th} *arm*

Initialize: $S_0 = \{1, 2, \ldots, n\}$

for $k = 0, 1, \ldots, \lceil \log_2(n) \rceil - 1$ **do**

  Pull each arm in S_k for $r_k = \lfloor \frac{B}{|S_k| \lceil \log_2(n) \rceil} \rfloor$ additional times and set $R_k = \sum_{j=0}^{k} r_j$

  Let σ_k be a bijection on S_k such that $\ell_{\sigma_k(1),R_k} \leq \ell_{\sigma_k(2),R_k} \leq \ldots \leq \ell_{\sigma_k(|S_k|),R_k}$

  $S_{k+1} = \{i \in S_k : \ell_{\sigma_k(i),R_k} \leq \ell_{\sigma_k(\lfloor |S_k|/2 \rfloor),R_k}\}$

end

Output: Singleton element of $S_{\lceil \log_2(n) \rceil}$

Jamieson and Talwalkar, *Non-stochastic Best Arm Identification and Hyperparameter Optimization*, 2016.

Successive halving

Number of initial configs, n=81. Total budget B=324

Halving rate $\gamma = 3$

k	$\lvert S_k \rvert$	r_k
0	81	1
1	27	3
2	9	9
3	3	27