Efficient algorithm design via automated algorithm selection and configuration

Alexander Tornede & Marius Lindauer

Euro PhD School Data Science Meets Combinatorial Optimisation

Program For Today

- 9:00 – 10:30
  Algorithm Selection

Coffee Break 

- 10:50 – 12:20
  Algorithm Configuration

Lunch Break 

- 15:00 – 16:30
  Algorithm Configuration & Hyperparameter Optimization Hands-on with SMAC

Interactive Sessions with Quizzes! 

Who are we? Alexander Tornede

- 2015/2018
  B.Sc/M.Sc. in Computer Science from Paderborn University
- 06/2023
  Defended Ph.D. in Computer Science on Machine Learning for Algorithm Selection at Paderborn University
- Since 09/2022:
  PostDoc of Marius' AutoML research group at Leibniz University Hannover
- Current research focus
  - Interactive and Explainable AutoML
  - LLMs for AutoML
  - (Uncertainty in AutoML)
- Hobbies:
  - Outdoor, sports, board games, computer games, reading

@ATornede

Website

Who are we? Marius Lindauer

- 2007/2010
  B.Sc./M.Sc. in Computer Science from Potsdam University
- 2015
  Defended Ph. D. in Computer Science on Automated Algorithm Selection, Schedules and Configuration at Potsdam University
- 2014–2019
  PostDoc in Frank Hutter’s lab at the University of Freiburg
- Since 2019
  Prof. of (Automated) Machine Learning at Leibniz University Hannover
- Current research focus
  - AutoML, Explainability, Reinforcement Learning, …
- Hobbies:
  - Go, Taekwondo, Computer Games

@LindauerMarius

Efficient algorithm design via automated algorithm selection and configuration

Alexander Tornede & Marius Lindauer

Euro PhD School Data Science Meets Combinatorial Optimisation

Session´s Story

- What is Algorithm Selection (AS)?
  - Motivation
  - Idea
  - Important Concepts
- Foundations of AS
  - Application Conditions
  - Instance Features
  - Loss Functions in AS
- Learning Selectors from Data
  - Desired Properties
  - Instantiations
- Latest Trends & Open Problems
  - Algorithm Features
  - Censored Data
  - Open Problems

What is Algorithm Selection?

Assume You Want to Sort An Array

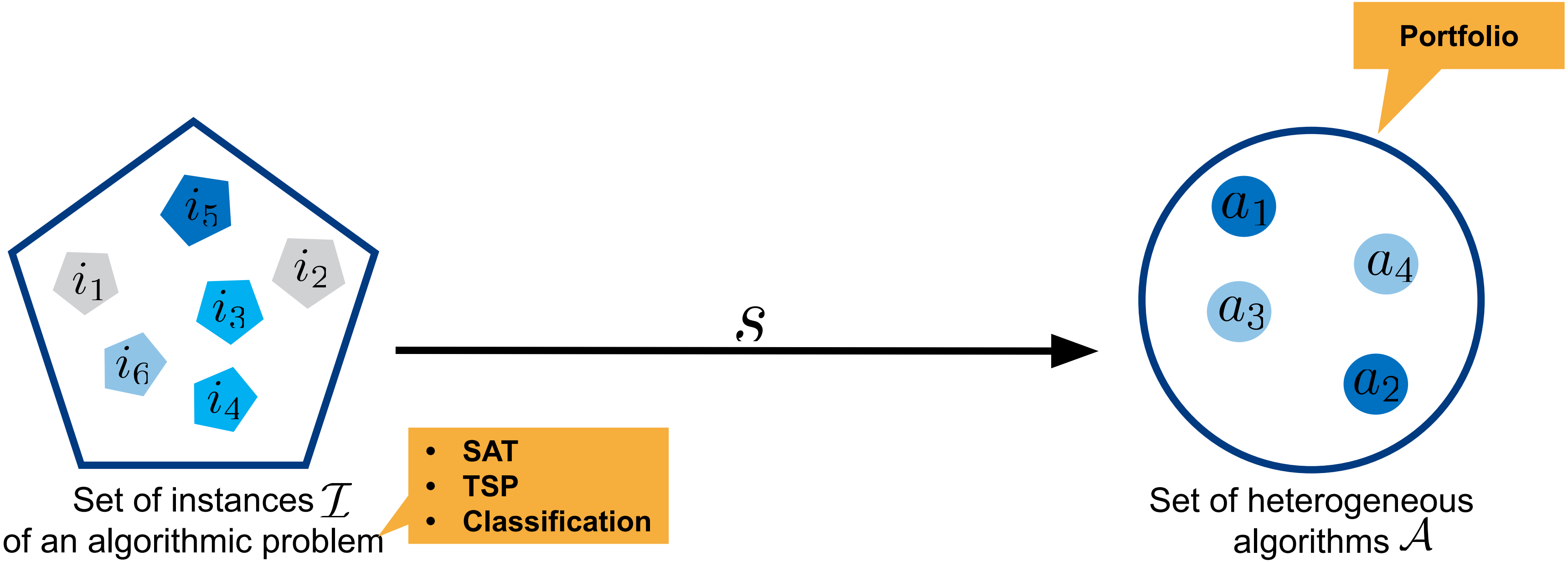
- Which algorithm would you choose and why? 🤔
- Would you always choose that algorithm?
- Can you think of an array where some algorithm might be faster than another?
  - Sorted array? → Insertion sort: O(n)

Comparison sorts

Name	Best	Average	Worst	Memory	Stable	Method	Other notes
Quicksort	$n \log n$	$n \log n$	n^2	$\log n$	No	Partitioning	Quicksort is usually done in-place with $O(\log n)$ stack space.[5][6]
Merge sort	$n \log n$	$n \log n$	$n \log n$	n	Yes	Merging	Highly parallelizable (up to $O(\log n)$ using the Three Hungarians' Algorithm).[7]
In-place merge sort	—	—	$n \log^2 n$	1	Yes	Merging	Can be implemented as a stable sort based on stable in-place merging.[8]
Introsort	$n \log n$	$n \log n$	$n \log n$	$\log n$	No	Partitioning & Selection	Used in several STL implementations.
Heapsort	$n \log n$	$n \log n$	$n \log n$	1	No	Selection	
Insertion sort	n	n^2	n^2	1	Yes	Insertion	$O(n + d)$, in the worst case over sequences that have *d* inversions.
Block sort	n	$n \log n$	$n \log n$	1	Yes	Insertion & Merging	Combine a block-based $O(n)$ in-place merge algorithm[9] with a bottom-up merge sort.
Timsort	n	$n \log n$	$n \log n$	n	Yes	Insertion & Merging	Makes *n-1* comparisons when the data is already sorted or reverse sorted.
Selection sort	n^2	n^2	n^2	1	No	Selection	Stable with $O(n)$ extra space, when using linked lists, or when made as a variant of Insertion Sort instead of swapping the two items.[10]
Cubesort	n	$n \log n$	$n \log n$	n	Yes	Insertion	Makes *n-1* comparisons when the data is already sorted or reverse sorted.
Shellsort	$n \log n$	$n^{4/3}$	$n^{3/2}$	1	No	Insertion	Small code size.
Bubble sort	n	n^2	n^2	1	Yes	Exchanging	Tiny code size.
Exchange sort	n^2	n^2	n^2	1	No	Exchanging	Tiny code size.
Tree sort	$n \log n$	$n \log n$	$n \log n$ (balanced)	n	Yes	Insertion	When using a self-balancing binary search tree.
Cycle sort	n^2	n^2	n^2	1	No	Selection	In-place with theoretically optimal number of writes.
Library sort	$n \log n$	$n \log n$	n^2	n	No	Insertion	Similar to a gapped insertion sort. It requires randomly permuting the input to warrant with-high-probability time bounds, which makes it not stable.
Patience sorting	n	$n \log n$	$n \log n$	n	No	Insertion & Selection	Finds all the longest increasing subsequences in $O(n \log n)$.
Smoothsort	n	$n \log n$	$n \log n$	1	No	Selection	An adaptive variant of heapsort based upon the Leonardo sequence rather than a traditional binary heap.
Strand sort	n	n^2	n^2	n	Yes	Selection	
Tournament sort	$n \log n$	$n \log n$	$n \log n$	n[11]	No	Selection	Variation of Heapsort.
Cocktail shaker sort	n	n^2	n^2	1	Yes	Exchanging	A variant of Bubblesort which deals well with small values at end of list
Comb sort	$n \log n$	n^2	n^2	1	No	Exchanging	Faster than bubble sort on average.
Gnome sort	n	n^2	n^2	1	Yes	Exchanging	Tiny code size.
Odd–even sort	n	n^2	n^2	1	Yes	Exchanging	Can be run on parallel processors easily.

Source: Wikipedia

The Algorithm Selection Problem [Rice, 1967]



Goal: For a given instance, choose algorithm which is optimal with respect to some loss function

$$\ell : \mathcal{I} \times \mathcal{A} \to \mathbb{R}$$

Runtime

Solving Algorithm Selection

- (Unknown) Oracle

$$s^*(i) = \arg\min_{a \in \mathcal{A}} \mathbb{E}[\ell(i,a)]$$

- Naive Solution: Exhaustive enumeration

$$s(i) = \arg\min_{a \in \mathcal{A}} \frac{1}{N} \sum_{n=1}^{N} \ell(i,a)$$

Costly to evaluate!

Solving AS: Surrogate Loss Functions

- Learn surrogate loss function based on training instances $\mathcal{I}_D$

Fast to evaluate!

$$\widehat{\ell} : \mathcal{I} \times \mathcal{A} \rightarrow \mathbb{R}$$

- Canonical algorithm selector

$$s(i) \in \arg\min_{a \in \mathcal{A}} \widehat{\ell}(i, a)$$

Represented by features

Static Selection: Single-Best Solver (SBS)

- Single best solver (SBS) always selects the algorithm best on average on the training data

$$\widehat{\ell}_{SBS}(i,a) = \frac{1}{|\mathcal{I}_D|} \sum_{i' \in \mathcal{I}_D} \ell(i', a)$$

a : an algorithm
i : an instance
$\mathcal{I}_D$: training instances
ℓ : original loss function
$\widehat{\ell}$: surrogate loss function

LUH|AI Leibniz Universität Hannover

Questions?

Kahoot Quiz 1: kahoot.it

Foundations of AS

When Can AS be Successfully Applied? 🤔

LUH|AI

Leibniz Universität Hannover

1. Multiple Algorithms Available

2. Performance Complementarity Among Algorithms

3. Availability of Instance Features

Instance Features

- Required to learn good surrogate loss functions from data 🤔
  - Generalization to unseen instances

- Need to fulfill certain requirements / desiderata

Instance Feature Properties

1. Correlation
   - Value of a feature should correlate with loss of an/multiple algorithm/s
2. Computation time
   - Fast to compute
3. Feature amount
   - Total amount should be as small as possible
4. Complementarity
   - Features should be complementary to each other in terms of their information
5. Domain independence
   - Feature should be ideally domain-independent

Types of Instance Features

Syntactic

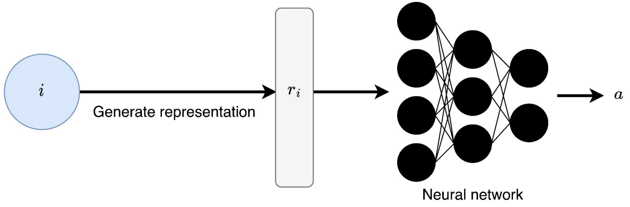
- Based on statistical properties of the instance
- Information extracted from structures of the instance
- Examples
  - number of decision variables
  - number of nodes of graph representation

Probing

- Extracted from the trajectory of a short run of an algorithm
- Examples
  - ELA features (blackbox optimization)
  - landmarkers (meta-learning)

Deep Learning Based

- Automatically learn complex features from an instance
- Examples
  - [Loreggia et al. 2016]
  - [Sigurdson et al. 2017]
  - [Sievers et al. 2019]



Types of Instance Features and Properties 🤔

Requirements	Instance feature kind		
	Syntactic	Probing	Deep learning-based
Correlation			
Computation time			
Feature amount			
Complementarity			
Domain independence			

Common AS Loss Functions

- We will distinguish loss functions tailored towards
  - Constraint **satisfaction** problems
    - Find **any** solution to the problem **quickly**

Time

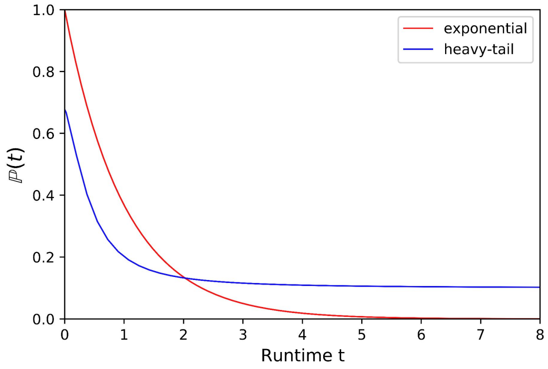
  - Constraint **optimization** problems
    - Find an **as-good-as-possible** solution the problem **quickly**

Quality

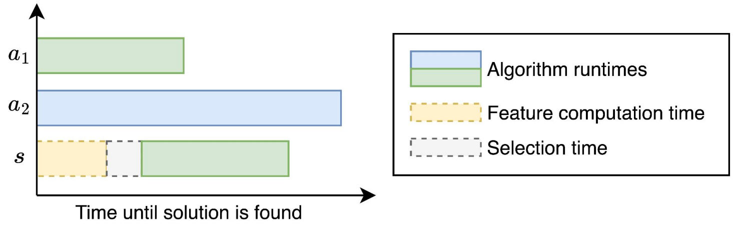
Time

Constraint Satisfaction Problems: Loss Functions (1)

- Time is of importance → Can we just focus on algorithm runtime? 🤔
  - No! **Time until solution is found** is more important!
- What happens if the algorithm does not find a solution (in our lifetime)? 🤔
  - Return without a solution
  - Takes extremely long



- But even if it finds a solution…



Constraint Satisfaction Problems: Loss Functions (2)

- Goal
  - Account for instances which we could not solve under a certain cutoff
  - Account for selection and feature computation time

- Solution
  - Penalized Average Runtime

$$\ell_{prK}(i,a) = \begin{cases} \ell_{runtime}(i,a) & \text{if } \ell_{runtime}(i,a) \leq C \\ K \cdot C & \text{else} \end{cases}$$

$$\mathcal{L}_{PARK}(I,s) = \frac{1}{|I|} \sum_{i \in I} \ell_{prK}(i, s(i))$$

a : an algorithm
i : an instance
$\mathcal{I}_D$: training instances
ℓ : original loss function
$\widehat{\ell}$: surrogate loss function

Problems of the ParK?

$$\ell_{prK}(i,a) = \begin{cases} \ell_{runtime}(i,a) & \text{if } \ell_{runtime}(i,a) \leq C \\ K \cdot C & \text{else} \end{cases}$$

1. Choice of K
   - Hard to make
   - Arbitrary
   - Larger K → large penalty for timeouts
   - How does a concrete requirement of a relative number of timeouts relate to a concrete K?

2. Hides a much more complicated underlying multi-objective problem
   - Very rough solution to the problem, but still SOTA

Constraint Optimization Problems: Loss Functions

- Solution quality

- E.g. a (inverse) machine learning loss function in case of machine learning as an algorithmic problem
  - accuracy
  - F1 score
  - etc.

Questions?

Kahoot Quiz 2: kahoot.it

Learning Selectors From Data

Learning a Selector / Surrogate Loss From Data

- Surrogate Loss Function

$$\widehat{\ell} : \mathcal{I} \times \mathcal{A} \to \mathbb{R}$$

Learn this!

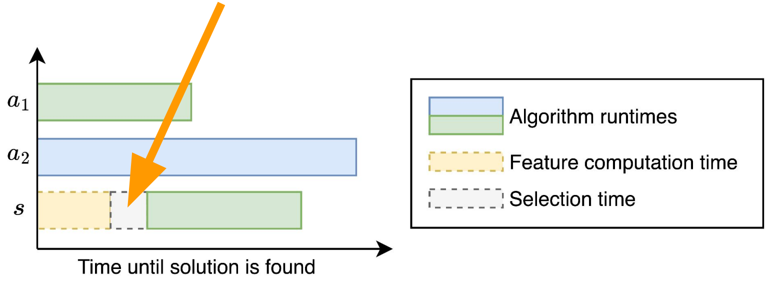
- Canonical algorithm selector

$$s(i) = \arg\min_{a \in \mathcal{A}} \widehat{\ell}(i, a)$$

a : an algorithm
i : an instance
$\mathcal{I}_D$: training instances
ℓ : original loss function
$\widehat{\ell}$: surrogate loss function

Desired Properties of a Surrogate Loss

1. Cheap to evaluate



2. Should mimic the original loss?

$$\forall i \in \mathcal{I}, a \in \mathcal{A} : \ell(i,a) \approx \widehat{\ell}(i,a)$$

Weaker: We want it to be **order-preserving**

$$\forall i \in \mathcal{I}, a_1, a_2 \in \mathcal{A} : \ell(i,a_1) \leq \ell(i,a_2) \Rightarrow \widehat{\ell}(i,a_1) \leq \widehat{\ell}(i,a_2)$$

a : an algorithm
i : an instance
$\mathcal{I}_D$: training instances
ℓ : original loss function
$\widehat{\ell}$: surrogate loss function

Order-Preserving Surrogate Losses

$$\forall i \in \mathcal{I}, a_1, a_2 \in \mathcal{A} : \ell(i, a_1) \leq \ell(i, a_2) \Rightarrow \widehat{\ell}(i, a_1) \leq \widehat{\ell}(i, a_2)$$

- Can we weaken that even more? 🤔
- If so, do we want to do that?

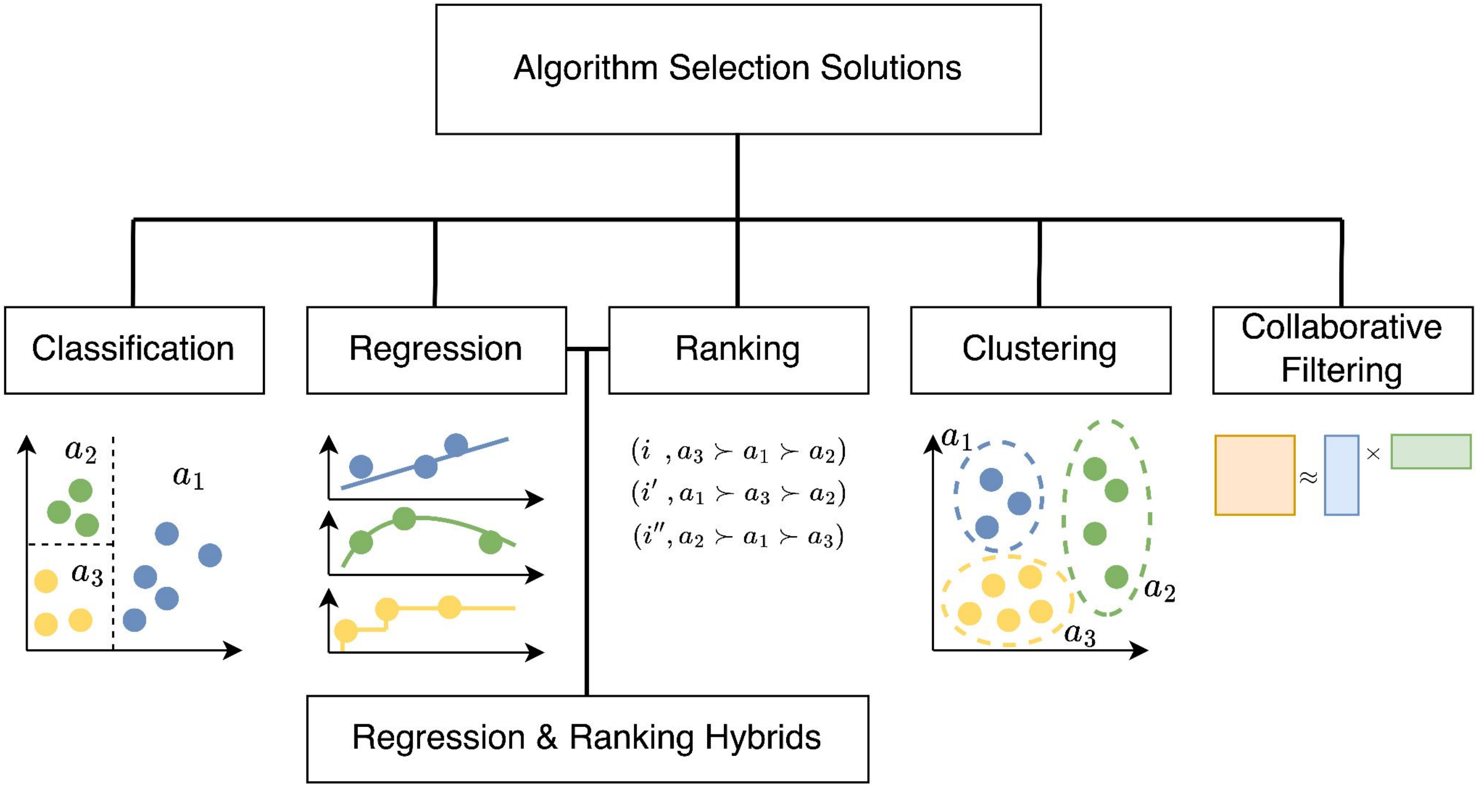
Desired Properties of Surrogate Losses

1. Cheap to evaluate
2. Order-preserving

- If we can fulfill these two properties on the complete instance space and for all algorithms, what does it entail for the selector? 🤔

$$s(i) \in \arg\min_{a \in \mathcal{A}} \widehat{\ell}(i, a)$$

Concrete Surrogate Loss Instantiations



Multi-Class Classification

- Surrogate loss

$$\widehat{\ell}_{classification}(i, a) = \begin{cases} 0 & \text{if } h(\boldsymbol{f}_i) = a \\ 1 & \text{else} \end{cases}$$

Multi-class classification model

$$h : \mathbb{R}^d \longrightarrow \mathcal{A}$$

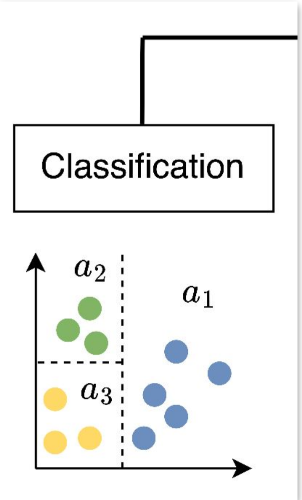
- Training data

$$\mathcal{D}_{classification} = \{(\boldsymbol{f}_i, a^*) | i \in \mathcal{I}_D \wedge \forall a \in \mathcal{A} : \ell(i, a^*) \leq l(i, a)\}$$

- Examples: [Guerri et al. 2004, Gent et al. 2010, Xu et al. 2011]

- Disadvantages?

SATzilla'11



SATzilla'11 [Xu et al. 2011]

- One-vs-one decomposition for multi-class classification
  - One binary classification model for each pair of algorithms

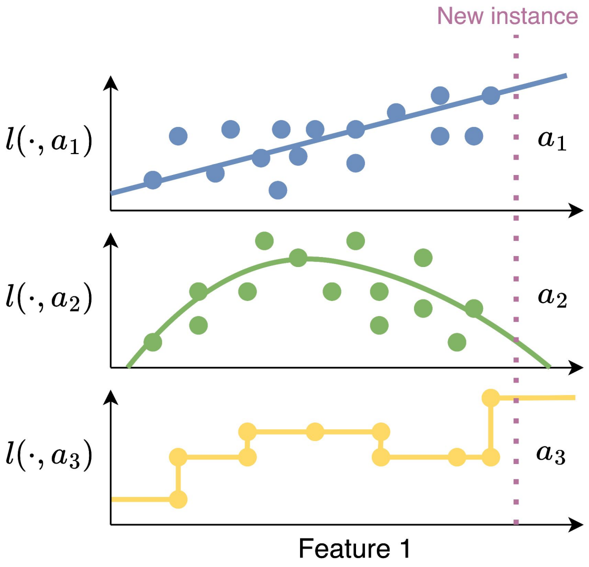
- Cost-sensitive classification
  - The more different two algorithms are in terms of their loss the higher the penalty for misclassification

Multi-Target Regression

- Multi-target regression problem where each algorithm's loss value is a regression target conditioned on the instance

- Often solved by decomposition into separate regression problems → one for each algorithm
  - Random forests are a common choice
  - Disadvantages? 🤔

- Examples: [Nudelman et al. 2004, Xu et al. 2008, Haim et al. 2009, Hutter et al. 2006]



Ranking

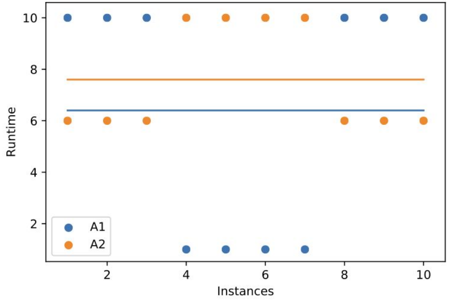
- Learn a model that does not concretely estimate the loss of each algorithm, but returns a ranking among these algorithms
- Select the highest ranked algorithm
- Can be modeled as a label ranking problem [Vembu et al. 2010]
- Disadvantages?

Desired Properties of Classification, Regression and Ranking? 🤔

1. Fast to compute
   - Holds for all

2. Order-preserving?
   - **Classification**:
     - No, at best top-1 preserving
   - **Regression**:
     - Yes, if solution is perfect.
     - Approximations can yield arbitrarily bad ranking performance
     - Actually much harder problem
   - **Ranking**:
     - Yes, but we might lose an idea of how close to algorithms are in terms of performance
     - Can yield arbitrarily bad regression performance

     → can be important for more sophisticated strategies



	a_1	a_2	a_3
GT	1.0	1.0	2.0
Sol 1	1.0	1.1	2.0
Sol 2	1.0	0.9	2.0

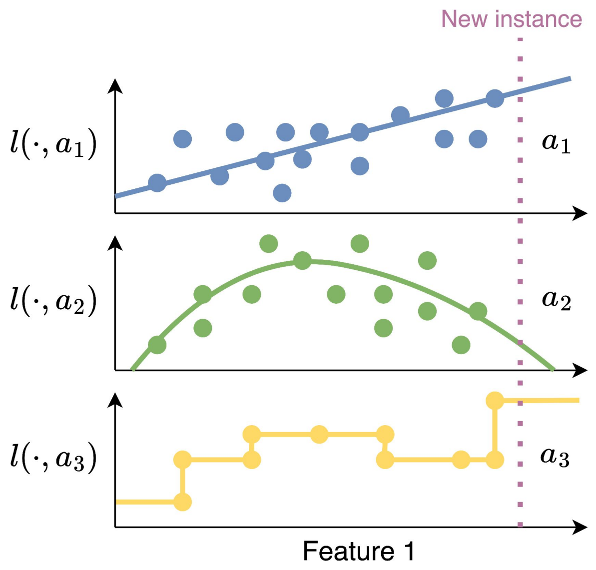
Questions?

Kahoot Quiz 3: kahoot.it

Latest Trends & Open Problems

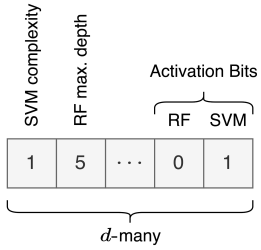
Disadvantages of Surrogate Decomposition 🤔

- Recall regression AS solution
  - Learn one regression model per algorithm
- Disadvantages?
  - Cannot exploit correlations between algorithms
  - Cannot handle unknown algorithms
  - Cannot account for algorithm behavior
- Solution?
  - Represent algorithms by features similar to instances!
    → **Learn one joint model across the joint feature space**!



Algorithm Features

- Should have similar properties as instance features
- Rather unstudied field so far
- Examples of works for algorithm features
  - [Tornede et al. 2022]: Use algorithm hyperparameters as features
  - [Pulatov et al. 2022]: Use source code features and control flow graph properties as features
  - [Cenikj et al. 2023]: Use time series features on the trajectory of the algorithm



Type	Name	Explanation	# Features
Code	Lines of code		2
	Cyclomatic complexity	number of independent execution paths [McCabe, 1976]	2
	Maxindent complexity	maximum level of indentation [Tornhill, 2018]	2
	Size of the sources		2
	Number of files		1
AST	Node count	number of nodes in the AST	1
	Edge count	number of edges in the AST	1
	Degrees of the nodes		5
	Transitivity	number of triangles on three connected nodes	1
	Clustering coefficient	measure the local connectivity of a node [Fagiolo, 2007]	4
	Depth	distance from the root node to each leaf	5
	Node type	based on Clang AST	6
	Edge type transition	based on Clang AST	36
	Operation type	data types operators are applied to	7
Dummy	ID	identifier of algorithm	1 per algorithm

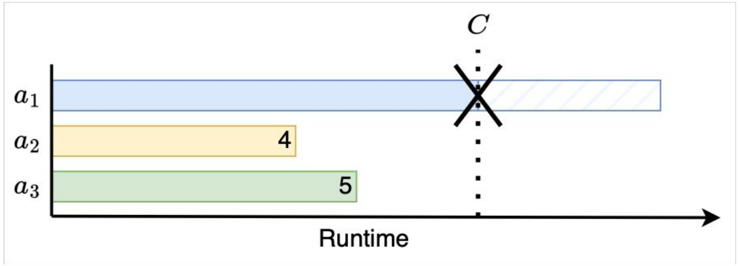
Table 2: Algorithm features considered in our study, grouped by type.

Source: [Pulatov et al. 2022]

Censored Data

- Why are some datapoints missing?
  → Timeouts!
- What to do with these samples?
  a. Drop the samples from the training data
  b. Impute the samples with
     - Cutoff
     - Multiple of cutoff
     - Mean
     - Etc. …

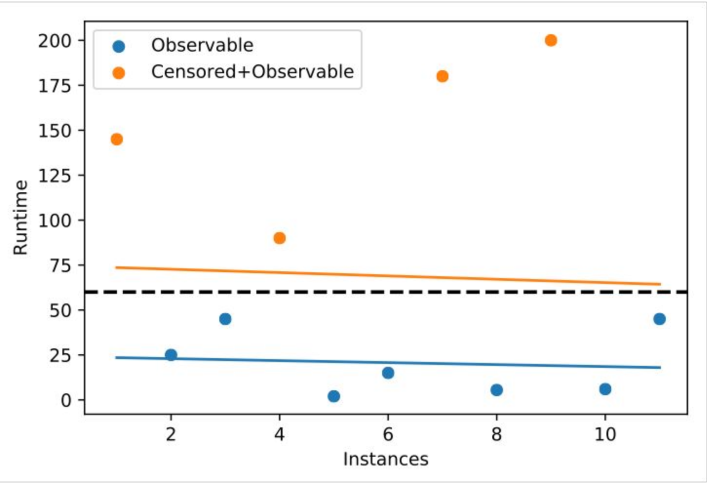
		a_1	a_2	a_3	…	…	…	…	…	…	…	a_{998}	a_{999}	a_{1000}
0.3, 2.7, …	i_1		0.16											
1.3, 5.3, …	i_2							0.91					0.34	
5.1, 6.7, …	…				0.86					0.24				
1.0, 0.0, …	…													
0.6, 1.9, …	i_{699}			0.38							0.78			
0.25, 2.27, …	i_{700}	0.01					0.67							



Dropping or Imputation?

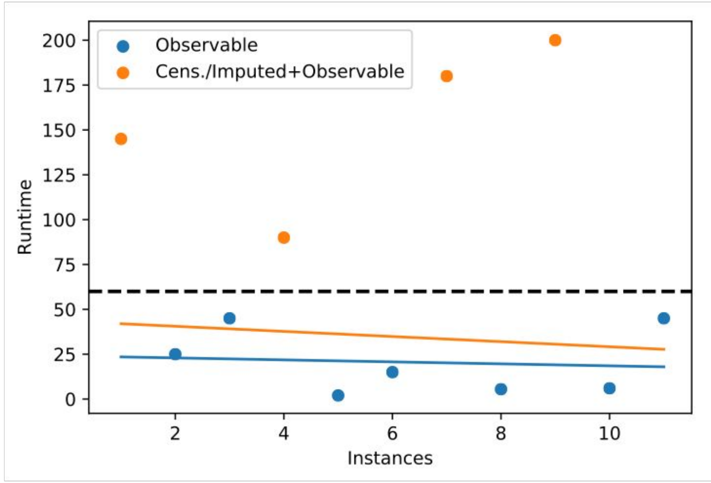
Dropping

- Systematic underestimation
  → Bad idea



Imputation with cutoff

- Systematic underestimation, but less severe than dropping
- Which imputation value to choose?



Survival Analysis (SA) to the Rescue

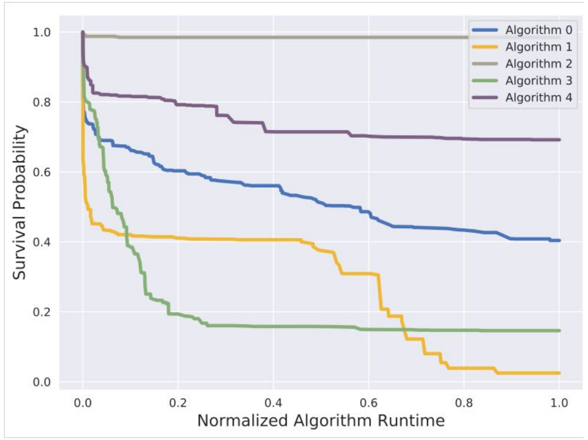
- Idea of Run2Survive[Tornede et al. 2020]
  - Model time until an algorithm stops as instance-dependent runtime / survival **distribution**
  - SA [Kleinbaum et al. 2012] can handle censored samples
- Learn a survival distribution for each algorithm

$$S_a(t, i) = \mathbb{P}(T_{a,i} \geq t | i)$$

- Choose algorithm with minimum decision theoretic expected loss

$$\arg\min_{a \in \mathcal{A}} \mathbb{E}[\mathcal{L}(T_{a,i})]$$

  - Expected runtime with identity as loss function
  - Do we always want the expected runtime?



Dangers of Expected Runtime

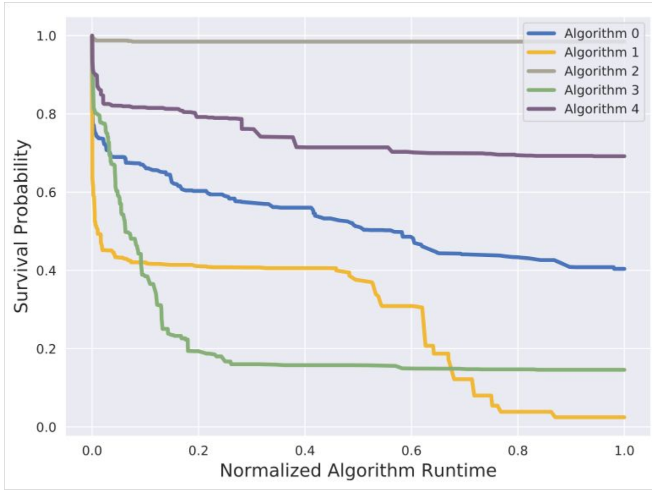
- Recall PARK loss

$$\ell_{prK}(i,a) = \begin{cases} \ell_{runtime}(i,a) & \text{if } \ell_{runtime}(i,a) \leq C \\ K \cdot C & \text{else} \end{cases}$$

- Which algorithm would you choose in case of a large K? 🤔

- Algorithm 3 vs Algorithm 1
  - Alg. 3 has lower expected runtime, but larger risk of timeout
  - Alg. 1 has larger expected runtime, but lower risk of timeout

- Solution: Risk-averse algorithm selection [Tornede et al. 2020]



Some Open Problems

- Hybrid ranking and regression models
  - First work by [Hanselle et al. 2020, Fehring et al. 2022]
- Transfer learning across problems
  - First work by [Deshpande et al. 2021]
- Grey-Box Algorithm Selection
  - (Besides algorithm features) first work by [Mohan et al. 2022, Ruhkopf et al. 2023]

Caveat: Many of these concepts are also explored in related fields such as algorithm configuration.

Questions?

Kahoot Quiz 4: kahoot.it

Further Material

- AS Survey [Kotthoff 2016]
- AS Survey [Kerschke et al. 2019]
- My dissertation :) [Tornede 2023]
- KI Campus Course “Automated Machine Learning”

Find Us

LUH|AI

Leibniz Universität Hannover

AutoML.org

@AutoML_org

automl

@AutoML_org

LUH|AI

@AIHannover

LUH-AI

@luh-ai

Funded by:

European Research Council
Established by the European Commission

DFG Deutsche Forschungsgemeinschaft

Federal Ministry of Education and Research

Federal Ministry for Economic Affairs and Energy

Federal Ministry for the Environment, Nature Conservation, Nuclear Safety and Consumer Protection

Niedersachsen