AutoML: Practical Considerations

Introduction

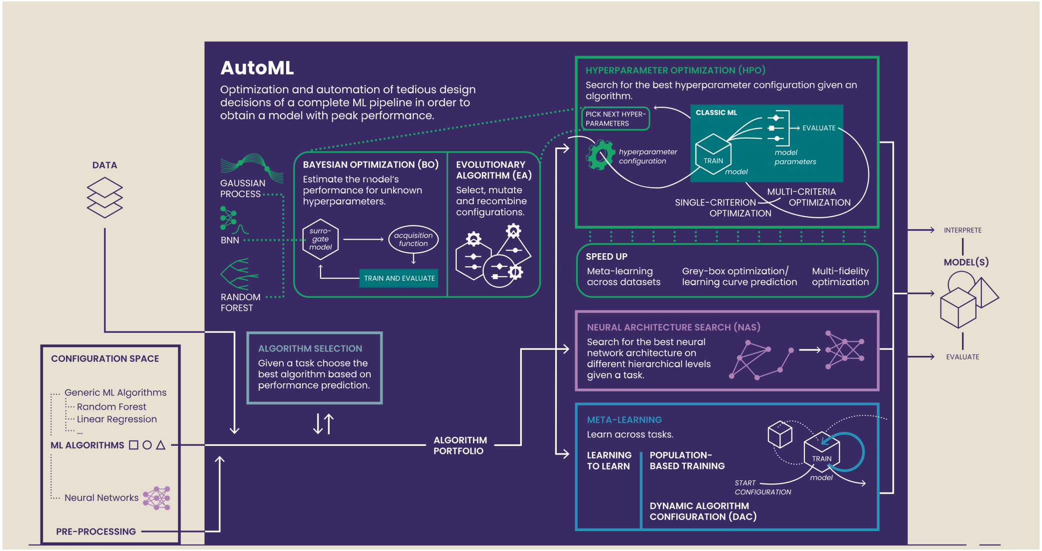
Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Janek Thomas Joaquin Vanschoren

From HPO to AutoML

So far we covered

- Mechanisms to select ML algorithms for a data set (algorithm selection)
- HPO as black-box optimization
  - Grid- and random search, EAs, BO
- HPO as a grey box problem
  - Hyperband, BOHB
- Neural Architecture Search (NAS)
  - One-Shot approaches, DART
- Dynamic algorithm configuration (learning to learn)
  - Adapt configuration during training

From HPO to AutoML


AutoML
Optimization and automation of tedious design decisions of a complete ML pipeline in order to obtain a model with peak performance.
DATA
GAUSSIAN PROCESS
BNN
RANDOM FOREST
BAYESIAN OPTIMIZATION (BO)
Estimate the model's performance for unknown hyperparameters.
surro-gate model
acquisition function
TRAIN AND EVALUATE
EVOLUTIONARY ALGORITHM (EA)
Select, mutate and recombine configurations.
HYPERPARAMETER OPTIMIZATION (HPO)
Search for the best hyperparameter configuration given an algorithm.
PICK NEXT HYPER-PARAMETERS
CLASSIC ML
EVALUATE
hyperparameter configuration
TRAIN
model
model parameters
SINGLE-CRITERION OPTIMIZATION
MULTI-CRITERIA OPTIMIZATION
SPEED UP
Meta-learning across datasets
Grey-box optimization/ learning curve prediction
Multi-fidelity optimization
NEURAL ARCHITECTURE SEARCH (NAS)
Search for the best neural network architecture on different hierarchical levels given a task.
META-LEARNING
Learn across tasks.
LEARNING TO LEARN
POPULATION-BASED TRAINING
TRAIN
model
START CONFIGURATION
DYNAMIC ALGORITHM CONFIGURATION (DAC)
ALGORITHM SELECTION
Given a task choose the best algorithm based on performance prediction.
ALGORITHM PORTFOLIO
CONFIGURATION SPACE
Generic ML Algorithms
Random Forest
Linear Regression
...
ML ALGORITHMS
Neural Networks
PRE-PROCESSING
INTERPRETE
MODEL(S)
EVALUATE

What is missing?

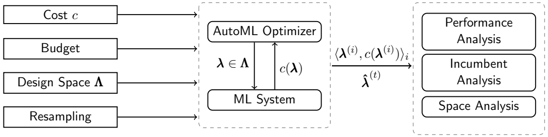
What do I need to know as an AutoML user?

- ~~Nothing, because it is automatic.~~
- Understand limitations of AutoML and framework.
- Know how to interpret the results.
- Maybe: Data cleaning and feature extraction.

Ingredients to implement an AutoML system?

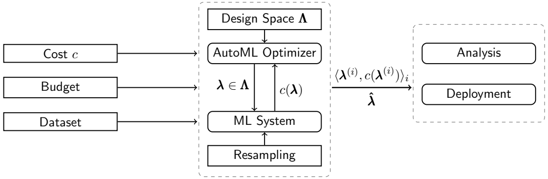
- HPO algorithm
- ML / Pipeline framework
- Parallelization / Multifidelity
- Process encapsulation and time capping

Academic view:



Cost c
Budget
Design Space $\boldsymbol{\Lambda}$
Resampling
AutoML Optimizer
$\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$
$c(\boldsymbol{\lambda})$
ML System
$\langle \boldsymbol{\lambda}^{(i)}, c(\boldsymbol{\lambda}^{(i)}) \rangle_i$
$\hat{\boldsymbol{\lambda}}^{(t)}$
Performance Analysis
Incumbent Analysis
Space Analysis

Practitioners view:



Cost c
Budget
Dataset
Design Space $\boldsymbol{\Lambda}$
AutoML Optimizer
$\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$
$c(\boldsymbol{\lambda})$
ML System
Resampling
$\langle \boldsymbol{\lambda}^{(i)}, c(\boldsymbol{\lambda}^{(i)}) \rangle_i$
$\hat{\boldsymbol{\lambda}}$
Analysis
Deployment

AutoML: Practical Considerations

Preprocessing

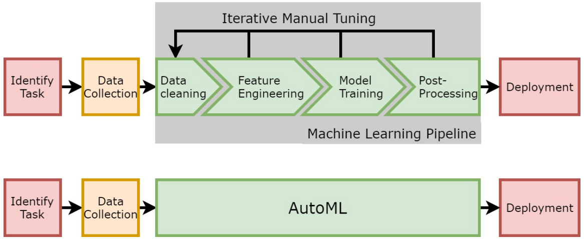
Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Janek Thomas Joaquin Vanschoren

Preprocessing

An AutoML systems should also optimize:

- ✗ Data preprocessing
- ✗ Feature engineering
- ✗ Feature selection
- ✓ Model training

✓ = already covered, ✗ = not covered so far



Simple Preprocessing and Cleaning

- Data cleaning is hard to fully automate since errors in the data can be semantic.
- A few simple things should always be done:
  - Remove ID columns, columns with only unique values
  - Remove duplicated columns
  - Remove constant columns
- Already harder, but still possible:
  - Detect outliers
  - Detect correct column types (numeric, discrete, datetime, ...)
  - Detect missing value encoding
- Even harder:
  - Detect spelling and formatting errors
  - Detect inconsistencies (e.g. zip code does not match city name)
  - Detect time series or spatial data

It is unclear how much of this is required input by the user (last point is more or less task specification) and what should be automated by the system.

Preprocessing: Categorical Features

Categorical features have a fixed number of distict (unordered) possible values called levels.

- For most learners categorical features need to be encoded in numeric features.
- Distinguish between binary, low cardinality and high cardinality categorical features. Low or high cardinality refers to the number of levels.
  - Binary: Encode as $1 - 0$.
  - Low-cardinality: One-hot encoding.
  - High-cardinality: Regularized target/impact encoding, clustering, hashing.
- Tree-based algorithms (in some software implementations) can natively handle even high-cardinality categorical features.
- Optimal encoding can vary between each feature, algorithm and hyperparameter configuration. Introduce and tune threshold hyperparameter that decides when to use high-cardinality encoding, e.g. $n_{lvls} \geq \tau_{\text{high card.}}$.
- The encoder should also be able to handle new feature levels occuring at test time without crashing.

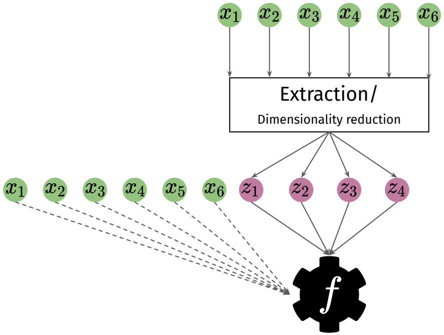
Preprocessing: Missing Values

Imputation is the process of replacing missing values with artificial substituted values.

- Simple imputation techniques replace missings with the mean, median, mode or a sample from empirical distribution of the feature.
- To keep track of the imputation, binary indicator features are added.
- For categorical features, missing values can easily be replaced by a new seperate level.
- Tree-based algorithms (in some software implementations) can natively handle missing values.
- Model-based imputation trains a machine learning model $f(x_{-i}) = x_i$ to predict missing values of x_i using the remaining features $x_{-i} = x_1, ..., x_{i-1}, x_{i+1}, ..., x_p$.
    - The imputation model should be able to handle missings natively.
    - The choice of learner and its hyperparameters add additional complexity to the imputation.
    - Random Forests are a reasonable choice.

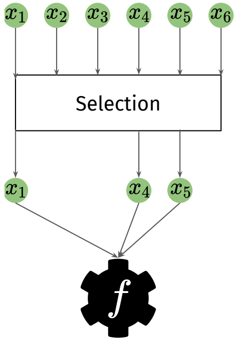
Feature Engineering

- $1 : 1$ Transformations
  - $\log$, $\sqrt{\cdot}$, ...
  - Standardization, Box-Cox, 0-1-scaling, ...
  - Binning of numeric features
- $1 : many$ Transformations
  - Polynomial expansion: $x_j \longrightarrow x_j, x_j^2, x_j^3, \dots$
  - Basis expansions with B/P-splines
  - Transform to "circular" features (month, day) e.g. $\tilde{x}_1 = sin(2\pi \cdot x_1/24)$
  - Extract Weekday/Day/... from datetime features
- $many : many$ Transformations
  - (kernel) Principal component analysis
  - Truncated singular value decomposition
  - Idependent component analysis
  - Autoencoder



Feature Selection

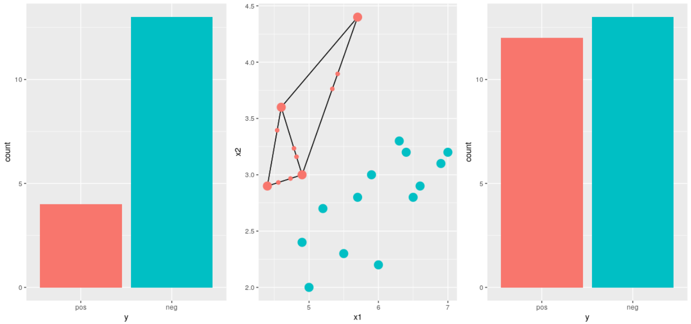
- Feature filter rank features and select most important fraction.
- Stepwise / wrapper methods greedily select one feature after another.
- Some learner have embedded feature selection CART, lasso, . . .
- p-dimensional bitvector as hyperparameters.
- Combined feature selection and HPO: [Binder et al. 2020]



Imbalanced Classes

Imbalanced classifcation occurs if one (usually more important) class appears much less frequent than other classes.

- Tune class weights as hyperparameters.
- Oversampling of minority class or undersmpling of majority class.
- Generate synthetic samples that (should) belong to the minority class, e.g. SMOTE.
- Be careful with resampling and ensure stratification.
- If imbalance is severe change framing, e.g. anomaly detection.



AutoML: Practical Considerations

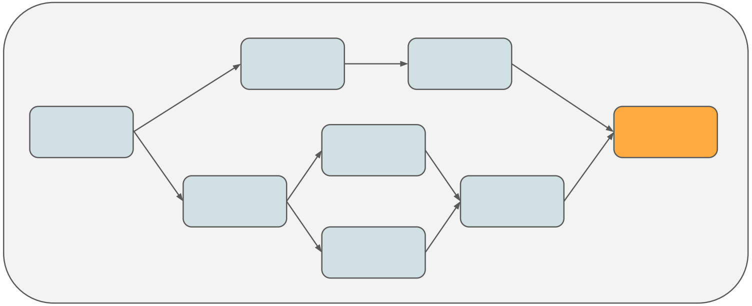
Machine Learning Pipelines

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Janek Thomas Joaquin Vanschoren

Pipelines and Workflows

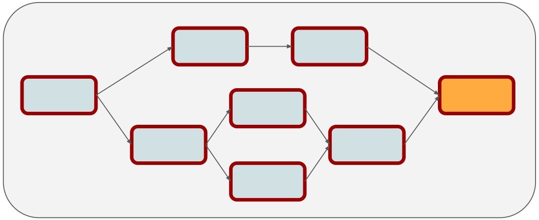
To build AutoML systems we need a language to describe:

- Machine learning algorithms
- Preprocessing operations
- Ensemble methodes like model averaging and stacking
- How data is passed from one stage to another

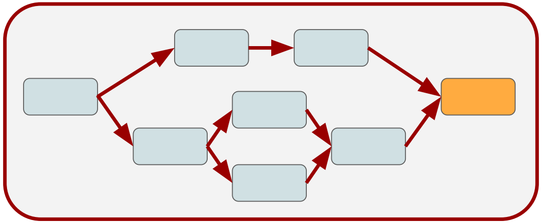


Pipelines and Workflows

Nodes: What is happening?



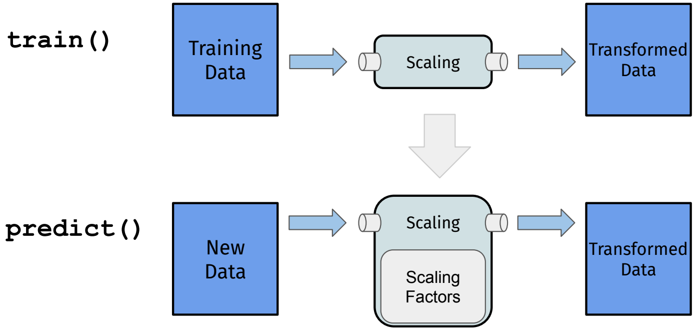
Edges: In what sequence is it happening?



$\longrightarrow$ We represent a ML workflow as a stateful directed Acyclic graph with nodes of operations and edged of data flow between them.

Training nodes

Each node can have *hyperparameters* $\mathbf{\Lambda}_{\text{node}}$, has to be *trained* and can *predict*.



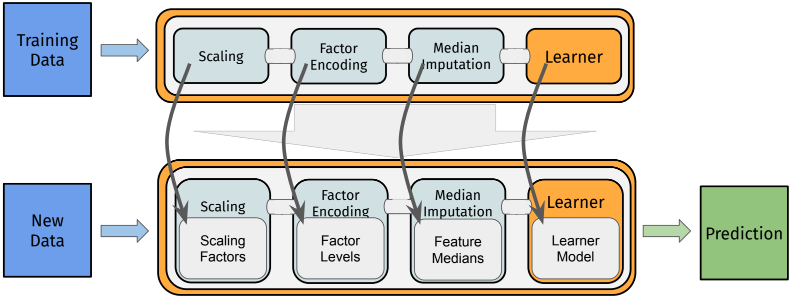
Transformed data can be

- a transformed version of the train/test data for preprocessing nodes, or
- predictions for learner nodes.

Linear Pipelines

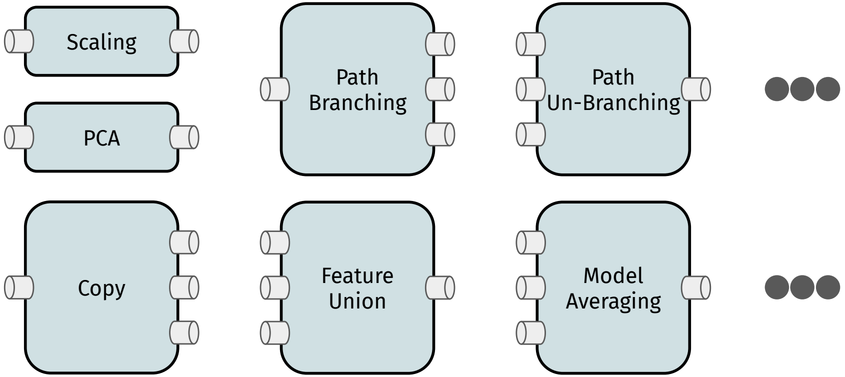
A linear pipeline behaves just as a machine learning algorithm.

- Can be evaluated with resampling and ensures that preprocessing does not cause data leakage.
- Hyperparameters $\mathbf{\Lambda} = \mathbf{\Lambda}_{\text{preproc}} \times \mathbf{\Lambda}_{\mathcal{I}}$ can be optimized jointly.



Nodes with Multiple Inputs or Outputs

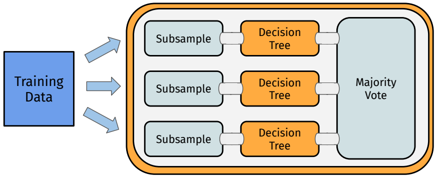
Nodes are not restricted to single inputs and single outputs.



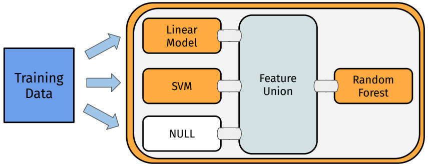
Ensemble Algorithms

This view allows easy representation of different ensemble algorithms:

Bagging:

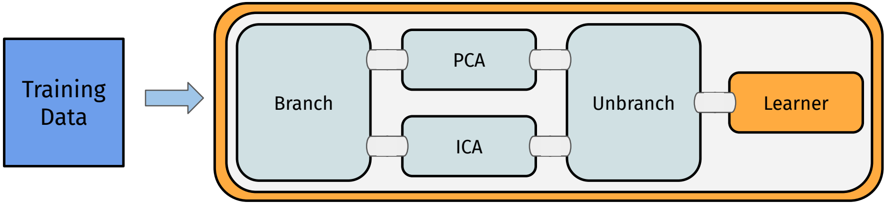


Stacking:

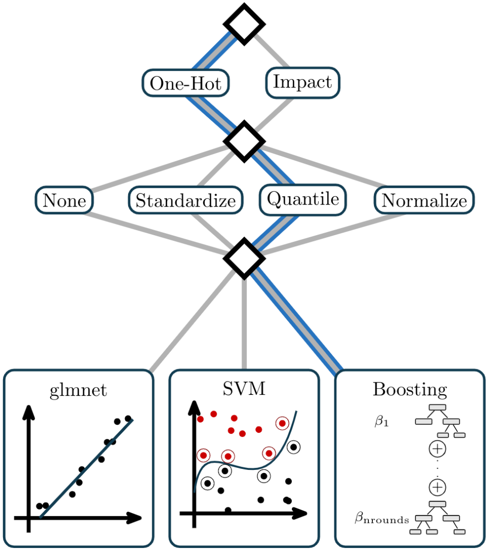


Branching

To represent the search space of an AutoML system there needs to be a branching node with a selection hyperparameter $\lambda_{\text{branch}} = (choice_1, ..., choice_k)$.



Example of a simple AutoML system



Search Space $\tilde{\Lambda}$

Name	Type	Bounds/Values	Trafo
◇ `encoding`	C	One-Hot, Impact	
◇ `scaling`	C	None, Standardize, Normalize, Quantile Transformation	
◇ `learner`	C	glmnet, SVM, Boosting	
`if learner = glmnet`			
`s`	R	$[-12, 12]$	2^x
`alpha`	R	$[0, 1]$	–
`if learner = SVM`			
`cost`	R	$[-12, 12]$	2^x
`gamma`	R	$[-12, 12]$	2^x
`if learner = Boosting`			
`eta`	R	$[-4, 0]$	10^x
`nrounds`	I	$\{1, \dots, 5000\}$	–
`max_depth`	I	$\{1, \dots, 20\}$	–

Pipeline Systems for Machine Learning Frameworks

Different frameworks to define and control such pipelines exist for most common programming languages:

- `scikit-learn` with `Pipeline`, `FeatureUnion` and `ColumnTransformer` classes for python.
- `mlr3` with `mlr3pipelines` extension for R.
- `ML.Net` for C#.
- `tfx` for tensorflow.
- `AutoMLPipeline` for Julia.
- ...

Each framework has slightly different features and limitations.

AutoML: Practical Considerations

Practical and Open Problems

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Janek Thomas Joaquin Vanschoren

Choice of Learning Algorithms

- A plethora of learners exists, for different data sets different models are likely needed.
- Studies and experience show:
  One these is often good – on tabular data:
  - Penalized regression, e.g. elastic net
  - Support vector machines
  - Gradient boosting
  - Random forests
  - Neural networks
- Example: Auto-Sklearn 2.0 [Feurer et al. 2020] uses:
  - Extra trees
  - Gradient boosting
  - Passive aggressive
  - Random forest
  - Linear regression with SGD
  - Multi-layer perceptron

Choice of Search Space for a Learning Algorithms

Algorithm	Hyperparameter	Type	Lower	Upper	Trafo
glmnet					
(Elastic net)	alpha	numeric	0	1	-
	lambda	numeric	-10	10	2^x
rpart					
(Decision tree)	cp	numeric	0	1	-
	maxdepth	integer	1	30	-
	minbucket	integer	1	60	-
	minsplit	integer	1	60	-
kknn					
(k-nearest neighbor)	k	integer	1	30	-
svm					
(Support vector machine)	kernel	discrete	-	-	-
	cost	numeric	-10	10	2^x
	gamma	numeric	-10	10	2^x
	degree	integer	2	5	-
ranger					
(Random forest)	num.trees	integer	1	2000	-
	replace	logical	-	-	-
	sample.fraction	numeric	0.1	1	-
	mtry	numeric	0	1	$x \cdot p$
	respect.unordered.factors	logical	-	-	-
	min.node.size	numeric	0	1	n^x
xgboost					
(Gradient boosting)	nrounds	integer	1	5000	-
	eta	numeric	-10	0	2^x
	subsample	numeric	0.1	1	-
	booster	discrete	-	-	-
	max_depth	integer	1	15	-
	min_child_weight	numeric	0	7	2^x
	colsample_bytree	numeric	0	1	-
	colsample_bylevel	numeric	0	1	-
	lambda	numeric	-10	10	2^x
	alpha	numeric	-10	10	2^x

Source: [Probst et al. 2019].

Ranges often selected based on experience

- See other AutoML frameworks: e.g. Auto-Sklearn 2.0 [Feurer et al. 2020]
- Sensitivity analysis often does not exist for ML algorithms
- Check literature on specific ML algorithm

Options for automation:

1. Use huge search space to cover all possibilities (combine with meta-learning for good initial design for Bayesian optimization)
    - Use results of meta-experiments to obtain smaller search space that is estimated to work well.
2. Start with a small space and increase bit by bit

Choice of Resampling Strategy

For computation of generalization error / cost:

$$c(\boldsymbol{\lambda}) = \frac{1}{k}\sum_{i=1}^{k} \widehat{GE}_{\mathcal{D}^i_{\text{val}}}\left(\mathcal{I}(\mathcal{D}^i_{\text{train}}, \boldsymbol{\lambda})\right)$$

Rules of thumb:

- Default: 10-fold CV ($k = 10$)
- Huge datasets: holdout
- Tiny datasets: 10x10 repeated CV
- Stratification for imbalanced classes

Watch out for this:

- Small sample size because of imbalances
- Repeated mesurements (leave-one-object out)
- Time dependencies
- A good AutoML system should let you customize resampling
- Meta-learn good resampling strategy [Feurer et al. 2020]

Choice of Optimization Algorithm

Choose optimization algorithm based on ...

- complexity of search space / budget
- time-costs of evaluations

Complex search space

$\rightarrow$ BO with RF surrogate, EA with exploratory character, TPE

Numerical (lower-dim) search space and tight budget

$\rightarrow$ BO with GP surrogate[1]

Expensive evaluations

$\rightarrow$ Hyperband, BOHB, DEHB

Deep learning

$\rightarrow$ common practice: Parameterize architectures, then HPO – better do it jointly!

$\rightarrow$ one-shot models and gradient-based optimization

[1]Still has its own hyperparameters [Lindauer et al. 2019]

Practical Problems: When to stop?

We need to specify a budget, e.g.

- walltime,
- function evaluations,
- performance threshold, or
- *stagnation* for a certain time.

Problems:

- Overtuning [Makarova et al. 2021].
- Missed opportunity.
- Wasted computational resources.

Ways out:

- Early stopping for BO [Makarova et al. 2021].
- Rules of thumb, maybe $50 \times l$ to $100 \times l$ (be careful and think for yourself!).
- Expert knowledge.

Practical Problems: Stability

AutoML system should:

- Never fail to return a result.
- Terminate within a given time.
- Save intermediate results and allow to continue.

Failure points:

- Optimizer can crash.
- Pipeline training can crash.
- Training of a pipeline can run "forever".

Ways out:

- Encapsulate train/predict in separate process from HPO.
- Ressource limit time and memory of that process.
- If pipeline crashes, run robust fallback (e.g., constant predictor).
- Use random configuration if optimizer crashes.

Practical Problems: Parallelization

Parallelization should allow:

- Multiple CPUs/GPUs on a single machine.
- Multiple machines / nodes.

Possible parallelization levels:

- Training of pipeline.
- Resampling.
- Evaluation of configurations (batch proposals or asynchronous).

Possible problems:

- Sequential nature of HPO algorithms (e.g. BO).
- Heterogeneous training times of pipelines can cause idling.
- Main memory or CPU-cache becomes bottleneck
- Communication between machine / nodes.

Way out: Use a robust framework for parallelization.

Practical Problems: What to return?

What is the output of a an AutoML system, e.g.

- Pipeline with best validation error.
- Stacking, e.g. averaging, of top-k pipelines.
- Pareto set for multi-objective optimization.
- "One-standard-error rule": Use the simplest model within one standard error of the performance of the best model [Hastie et al. 2009].

Ensure that simple but efficient pipelines have been tried out

- Baseline (Classification: Majority vote; Regression: Mean prediction).
- Linear Model.
- (untuned) Random Forest.
- ...

Open Problems

- Most efficient HPO approach? Good benchmarks often missing.
- How to integrate human a-priori knowledge?
- Human-in-the-loop approaches for AutoML.
- How can we best (computationally) transfer "experience" into AutoML?
- Warmstarts, learned search spaces, etc.
- Multi-Objective goals, including model intepretability and fairness.
- AutoML as a process is too much of a black-box, hurts adoption.
- Incorporate Uncertainty quantification into AutoML.
- AutoML beyond supervised learning.
- ...

$\longrightarrow$ Lots of open research questions, feel free to approach us for if you are interested.