AutoML: Hyperparameter Optimization

Wrap Up

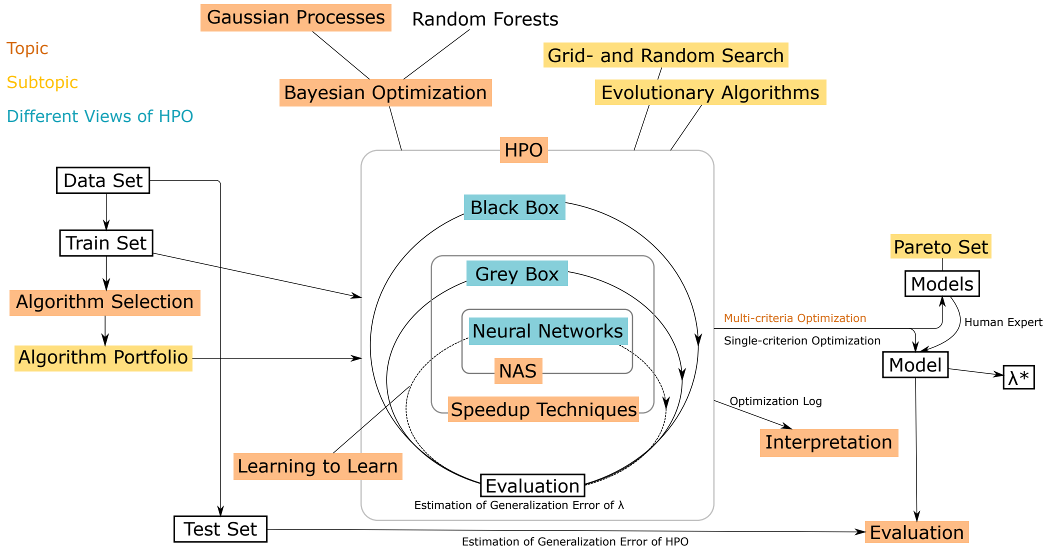
Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Jakob Richter

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



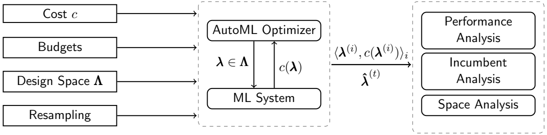
Lecture Overview

What is missing?

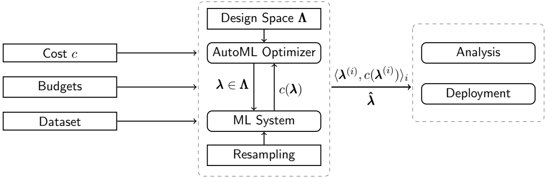
What do I need to know as an AutoML user?

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

Academic view:



Practitioners view:



Ingredients to implement an AutoML?

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

Choice of Learning Algorithm

- A plethora of learners exists, for different data sets different models are likely needed.
- Studies [Fernandez-Delgado et al. 2014] and experience show:
  One these is often good – on tabular data:
  - penalized regression
  - SVM
  - 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 model

Choice of Search Space for a Learning Algorithm

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:

- 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.

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

Taken from [Probst et al. 2019].

Choice of Search Space for a Learning Algorithm

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:

- 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.

Parameter	Def.P	Def.O	Tun.P	Tun.O	$q_{0.05}$	$q_{0.95}$
glmnet			0.069	0.024		
alpha	1	0.403	0.038	0.006	0.009	0.981
lambda	0	0.004	0.034	0.021	0.001	0.147
rpart			0.038	0.012		
cp	0.01	0	0.025	0.002	0	0.008
maxdepth	30	21	0.004	0.002	12.1	27
minbucket	7	12	0.005	0.006	3.85	41.6
minsplit	20	24	0.004	0.004	5	49.15
kknn			0.031	0.006		
k	7	30	0.031	0.006	9.95	30
svm			0.056	0.042		
kernel	radial	radial	0.030	0.024		
cost	1	682.478	0.016	0.006	0.002	920.582
gamma	$1/p$	0.005	0.030	0.022	0.003	18.195
degree	3	3	0.008	0.014	2	4
ranger			0.010	0.006		
num.trees	500	983	0.001	0.001	206.35	1740.15
replace	TRUE	FALSE	0.002	0.001		
sample.fraction	1	0.703	0.004	0.002	0.323	0.974
mtry	$\sqrt{p}$	$p \cdot 0.257$	0.006	0.003	0.035	0.692
respect.unordered.factors	TRUE	FALSE	0.000	0.000		
min.node.size	1	1	0.001	0.001	0.007	0.513
xgboost			0.043	0.014		
nrounds	500	4168	0.004	0.002	920.7	4550.95
eta	0.3	0.018	0.006	0.005	0.002	0.355
subsample	1	0.839	0.004	0.002	0.545	0.958
booster	gbtree	gbtree	0.015	0.008		
max_depth	6	13	0.001	0.001	5.6	14
min_child_weight	1	2.06	0.008	0.002	1.295	6.984
colsample_bytree	1	0.752	0.006	0.001	0.419	0.864
colsample_bylevel	1	0.585	0.008	0.001	0.335	0.886
lambda	1	0.982	0.003	0.002	0.008	29.755
alpha	1	1.113	0.003	0.002	0.002	6.105

Table 3: Defaults (package defaults (Def.P) and optimal defaults (Def.O)), tunability of the hyperparameters with the package defaults (Tun.P) and our optimal defaults (Tun.O) as reference and tuning space quantiles ($q_{0.05}$ and $q_{0.95}$) for different parameters of the algorithms.

Taken from [Probst et al. 2019].

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 situation because of imbalancies
- 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

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

Numerical (lower-dim) search space and tight budget

$\rightarrow$ BO with GP

Expensive evals

$\rightarrow$ Hyperband, BOHB

Deep learning

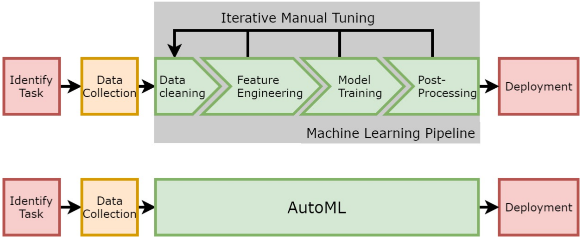
$\rightarrow$ Parametrize architectures, then HPO, see above

$\rightarrow$ NAS

Preprocessing

Ideal AutoML systems should also optimize:

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



✓ = already covered, ✗ = not covered so far

Lecture Overview

Preprocessing capabilities differ heavily

Tool	Platform	Input data sources		Data pre-processing	Data types detected					Feature engineering				ML Tasks		Model selection and Hyperparameter optimization					Quick start / early stop			Model evaluation / Result analysis/ Visualization		
		Spreadsheet datasets	Image, text		Numerical	Categorical	Datetime	Time-series	Other (Hierarchical types) (7*)	Datetime, categorical processing	Imbalance, missing values	Feature selection, reduction	Advanced feature extraction (8*)	Supervised learning (9*)	Unsupervised learning (10*)	Ensemble	Genetic algorithm	Random search	Bayesian search	Neural architecture search	Quick finding of starting model	Allow maximum limit search time	Restrict time consuming combination of components	Model dashboard	Feature importance	Model explainability and interpretation, and reason code (11*)
TransmogrifAI	Apache Spark	Y	N	Y(*)	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y	N	N			Y	Y	
H2O-AutoML	H2O clusters	Y	N	Y	Y	Y	Y	Y	N	Y	Y	Y	N	Y	N	Y	N	Y	N	N	N	Y	Y	Y	Y	Y
Darwin (+)	GCP	Y	N	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	N	Y	Y	Y	N	Y	Y	Y
DataRobot (+)	Datarobot & AWS	Y	Y	Y	Y	Y	N	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	Y		Y	Y	Y
Google AutoML (+)	Google Cloud	N	Y	Y						N	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Auto-sklearn		Y	N	N	N	N	N	N	N	Y(2*)	Y	Y	Y	Y	N	Y	N	Y	Y	N	Y	Y	Y	Y	Y	Y
MLjar (+)	MLJAR Cloud	Y(3*)	N	Y	Y	Y	N	N	N	Y	Y(4*)	N	N	Y(5*)	N	Y	N	Y	N	N	N	N	N	Y	Y	N
Auto_ml		Y	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	N	Y	N	Y	Y	N	N	N	N	Y	Y	Y
TPOT		Y	N	N	N	N	N	N	N	N	Y	N	Y	Y	N	Y	Y	N	N	N	N	Y	N	Y	Y	N
Auto-keras		Y	Y	N	N	N	N	N	N	N	Y	Y	N	Y	N	N	N	Y	Y	Y	Y	Y	N	Y	N	Y
Ludwig		Y	Y	Y(*)	Y	Y	N	Y	Y	N	Y	Y	Y	Y	N	Y	N	Y	Y	Y	Y	N	N	Y	Y	N
Auto-Weka		Y	N	N	Y	Y	N	N	N	N	Y	Y	N	Y	N	Y	N	Y	Y	N	N	Y	Y	Y	N	N
Azure ML (+)	Azure	Y	Y	Y(6*)	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	N	Y	N	Y	Y	N		Y	Y	Y	Y	
Sagemaker (+)	AWS	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	N		Y	N	Y	Y	Y
H2O-Driverless AI (+)	H2O clusters	Y(3*)	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	Y	Y	Y	Y

Fig. 2. Comparison table of functionality for AutoML tools. (+): commercialized tools; (∗): the function is not very stable, it fails for some datasets; (2∗): categorical input must be converted into integers; (3∗): datasets have to include headers; (4∗): missing values must be represented as NA; (5∗): multiclass classification not provided; (6∗): need some users' input for dataset description such as column types; (7∗): ability to detect primitive data types and rich data types such as: text (id, url, phone), numerical (integer, real); (8∗): advanced feature processing: bucketing of values, removing features with zero variance or features with drift over time; (9∗): supervised learning includes binary classification, multiclass classification, regression; (10∗): unsupervised learning includes clustering and anomaly detection; (11∗): model interpretation and explainability refers to techniques such as LIME, Shapley, Decision Tree Surrogate, Partial Dependence, Individual Conditional Expectation, Lift chart, feature fit, prediction distribution plot, accuracy over time, hot spot and reason codes; In a few empty cells, it is not clear that the functionality is provided from documentations of the tools, to the best of our knowledge.

Taken from [Truong et al. 2019].

Highlighted: Non-commercial AutoML frameworks

- Auto-detection of feature types: some
- Preprocess categoricals: some
- Imputation: all
- Class imbalance handling: all

Cleaning

Data cleaning is hard to fully automate, but a few heuristics exist:

- Remove ID columns, columns with mostly unique values.
- Outlier detection
- Detect time series or spatial data $\rightarrow$ at the very least, evaluation and resampling needs to be adapted, but feature extraction likely as well

It is highly 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.

Categorical Features: Dummy Encoding

- Most simple encoding
- Works well with low cardinality categoricals

SalePrice	Central.Air	Bldg.Type
189900	Y	1Fam
195500	Y	1Fam
213500	Y	TwnhsE
191500	Y	TwnhsE
236500	Y	TwnhsE

SalePrice	Y	Bldg.Type.2fmCon	Bldg.Type.Duplex	Bldg.Type.Twnhs	Bldg.Type.TwnhsE
189900	1	0	0	0	0
195500	1	0	0	0	0
213500	1	0	0	0	1
191500	1	0	0	0	1
236500	1	0	0	0	1

Categorical Features: Target / Impact Encoding

- Well known from CART
- Handles high cardinality categoricals, too

Goal: Encodes each categorical $\boldsymbol{x}$ as a single numeric $\tilde{\boldsymbol{x}}$

Regression: $\mathrm{Impact}(x) = \mathbb{E}(\boldsymbol{y}|x) - \mathbb{E}(\boldsymbol{y})$

Classification: $\mathrm{Impact}(x) = \mathrm{logit}(P(y = \text{target}|x)) - \mathrm{logit}(P(y = \text{target}))$

- Needs regularization (through CV) to prevent target leakage [Zumel et al. 2019]
- Advantage: Handles unknown categorical levels on test data

Alternatives:

- factorization machines
- clustering feature levels
- feature hashing

$\mathcal{D}_{\text{train}}$

x	y
A	10
A	20
B	30

Encoding

x	$\tilde{x}$
A	-5
B	10

$\mathcal{D}_{\text{test}}$

x	$\tilde{x}$
A	-5
B	10
C	0

Common Preprocessing Steps: Missing Values

- Replace missings with imputed values, try not to disturb distribution too much
- Examples: mean, median, mode, sample from histogram, predict value based on other features
- Additional factor column indicating missingness can help if NA-state carries information for target
- *Out-Of-Range* works often well for tree-based techniques (RF and boosting!), saves extra missingness col
- For inference, simple imputation is often shunned, and multiple, model-based imputation recommended; for prediction naive imputation often works remarkably well

$\mathcal{D}_{\text{train}}$

x_1	x_2
10	3
NA	2
5	NA

Feature	NA
x_1	100
x_2	30

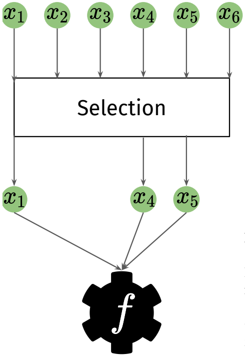
$\tilde{\mathcal{D}}_{\text{train}}$

x_1	x_2
10	3
100	2
5	30

Out of range imputation

Feature Selection

- Filter; for ultra-highdim tasks
- Stepwise / wrapper methods; seldom increases performance when well-regularized learners are used, but quite expensive
- Embedded: CART, lasso, ...
- Selection interesting when predictive performance vs. sparseness should be optimized $\rightarrow$ multi-criteria optimization
- Combined feature selection and HPO: [Binder et al. 2020]



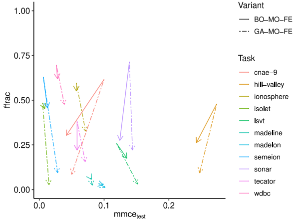


Figure 3: Comparison of multi-objective and single-objective methods applied to the SVM learning algorithm: Performance $\mathrm{mmce}_{\mathrm{test}}$ and the fraction of included features ffrac found after 2000 evaluations by baseline BO-SO (tail end of arrows), BO-MO-FE (head of solid arrows) and GA-MO-FE (head of dashed arrows). Each dataset (Table 1) is shown, values are averaged over 10 outer CV runs. Choice of individuals for MO methods described in Section 6.2.

Taken from [Binder et al. 2020].

Common Feature Construction Methods

Reduction:

- PCA, ICA, autoencoder

Feature extraction:

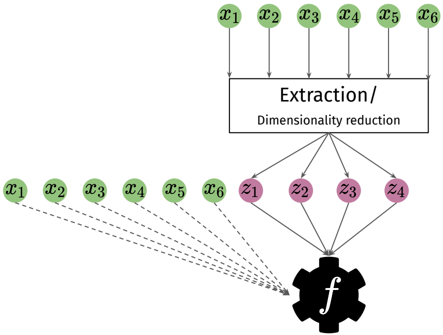
- Polynomial features: $x_j \longrightarrow x_j, x_j^2, x_j^3, \ldots$
- Interactions: $x_j, x_k \longrightarrow x_j, x_k, x_j \cdot x_k$

Feature generation:

- Transform to "circular" features (month, day)
  e.g. $\tilde{x}_1 = sin(2\pi \cdot x/24)$ and $\tilde{x}_2 = cos(2\pi \cdot x/24)$

Combine with external data:

- names $\longrightarrow$ gender, ethnicity, age
- home address $\longrightarrow$ household income
- location + date $\longrightarrow$ weather



Imbalanced Classes

- Oversampling of minority class
- Seldom: undersampling of majority class
- Intelligent oversampling strategies which create synthetic observations (SMOTE)

images/smote.png

Lecture Overview

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 (e.g. GP in BO)
- training can crash (e.g. segfault)
- training can run "forever"

Ways out

- Encapsulate train/predict in separate process from HPO
- Ressource limit time and memory of that process by OS
- If learner crashes, run robust fallback (constant predictor)
- Use "robust" HPO, run random config as last resort if proposal fails (exploration)

Practical Problems: Parallelization

Parallelization should allow:

- multiple cores
- multiple nodes

Possible parallelization levels:

- training of learner (threading / GPU)
- resampling
- eval configurations (batch proposal of HPO)

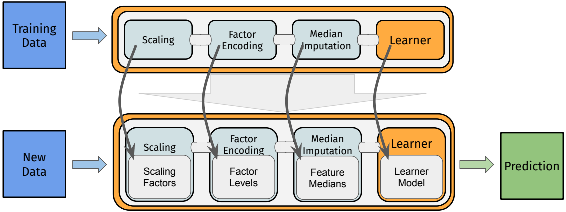
Possible problems:

- Sequential nature of HPO algorithms (e.g. BO)
- Heterogeneous runtimes cause idling $\longrightarrow$ asynchronous HPO attractive, but more complex
- Main memory or CPU-cache becomes bottleneck
- Communication between workers

Lecture Overview

Linear Pipelining

- Applying preprocessing to the whole dataset leads to data leakage
- Preprocessing should have train and predict steps, too
- Can add it to learner, and embed it in CV
- Surprise: Preproc has hyperparameters
- Optimize pipeline jointly:
  $\mathbf{\Lambda} = \mathbf{\Lambda}_{\text{preproc}} \times \mathbf{\Lambda}_{\mathcal{I}}$
- Still HPO, not much different than for single learner
- $\mathbf{\Lambda}$ "simply" of higher dimension



Nonlinear Pipelines

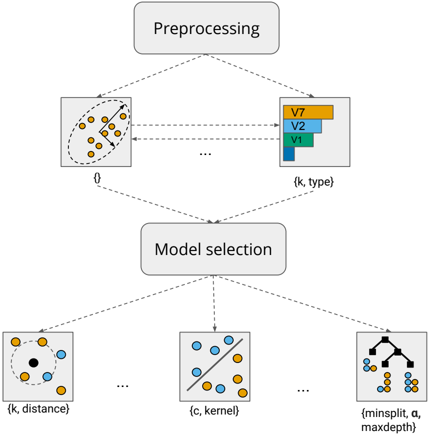
Ideal to let HPO choose automatically:

- preprocessing
- feature extraction
- learner

$\rightarrow$ $\boldsymbol{\Lambda}$ becomes hierarchical search space!

Suitable optimizers:

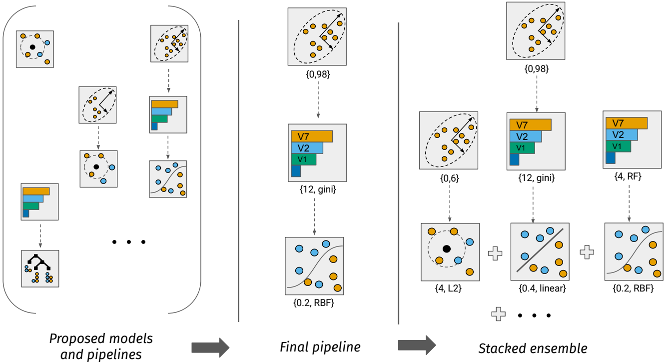
- TPE
- Random search, hyperband with sampler that follows the hierarchy
- BO with RF (imputation or hierarchical trees)
- Evolutionary approaches (similar to NAS)



Obtaining Final Model

Options:

- Choose the optimal path as linear pipeline.
- Build ensemble of best configurations
  (e.g. [Feurer et al., NIPS 2015], [LeDell and Poirier. 2020]).



Current Benchmark on Tabular Data

images/gijsbers_open_2019_tab2.pdf

Table taken from [Gijsbers et al., 2019].

- On some datasets AutoML yields big performance improvements
- On many datasets RF is equally good
- Need more and diverse benchmarks