Automated Machine Learning: A Partical Introduction

KonKIS’24 - Conference of the German AI Service Centers

Rise of Literacy

Photo by Anna Hunko on Unsplash

- Only priests were able to read and write
- People believed that they don’t need to read and write
- They went to the holy buildings

- Today, everyone can read and write
- No one doubts the benefits of it

- ⇒ **Democratization of literacy**

Inspired by Andrew Ng

Rise of AI Literacy?

Photo by Max Duzij on Unsplash

- Only highly educated people can program new AI applications
- Power lies only with the large IT companies

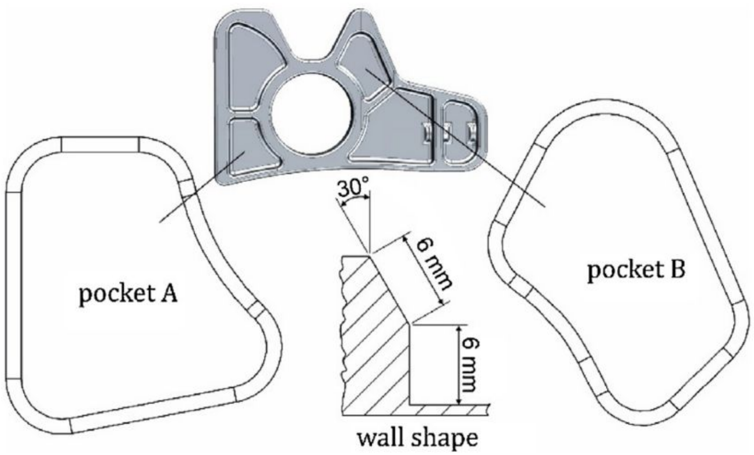
- In an age of limited resources, the need for efficient use gets more important

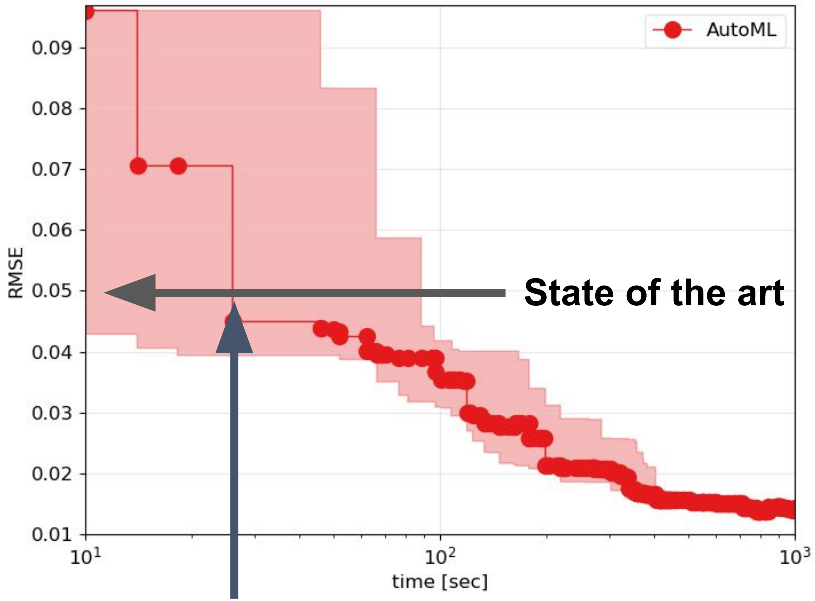
- **AutoML contributes to AI literacy!**

[See also my TEDx Talk]

Shape Error Prediction in Milling Process

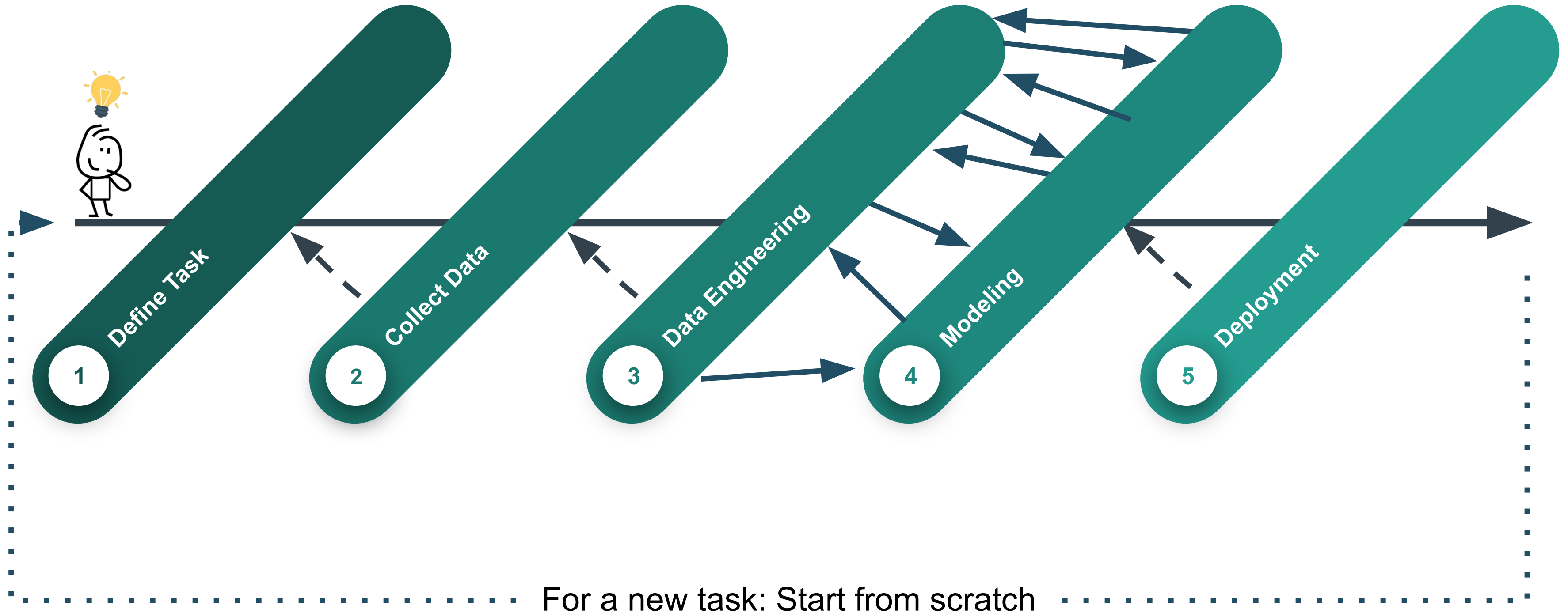
[Denkena et al. SSRN'20]



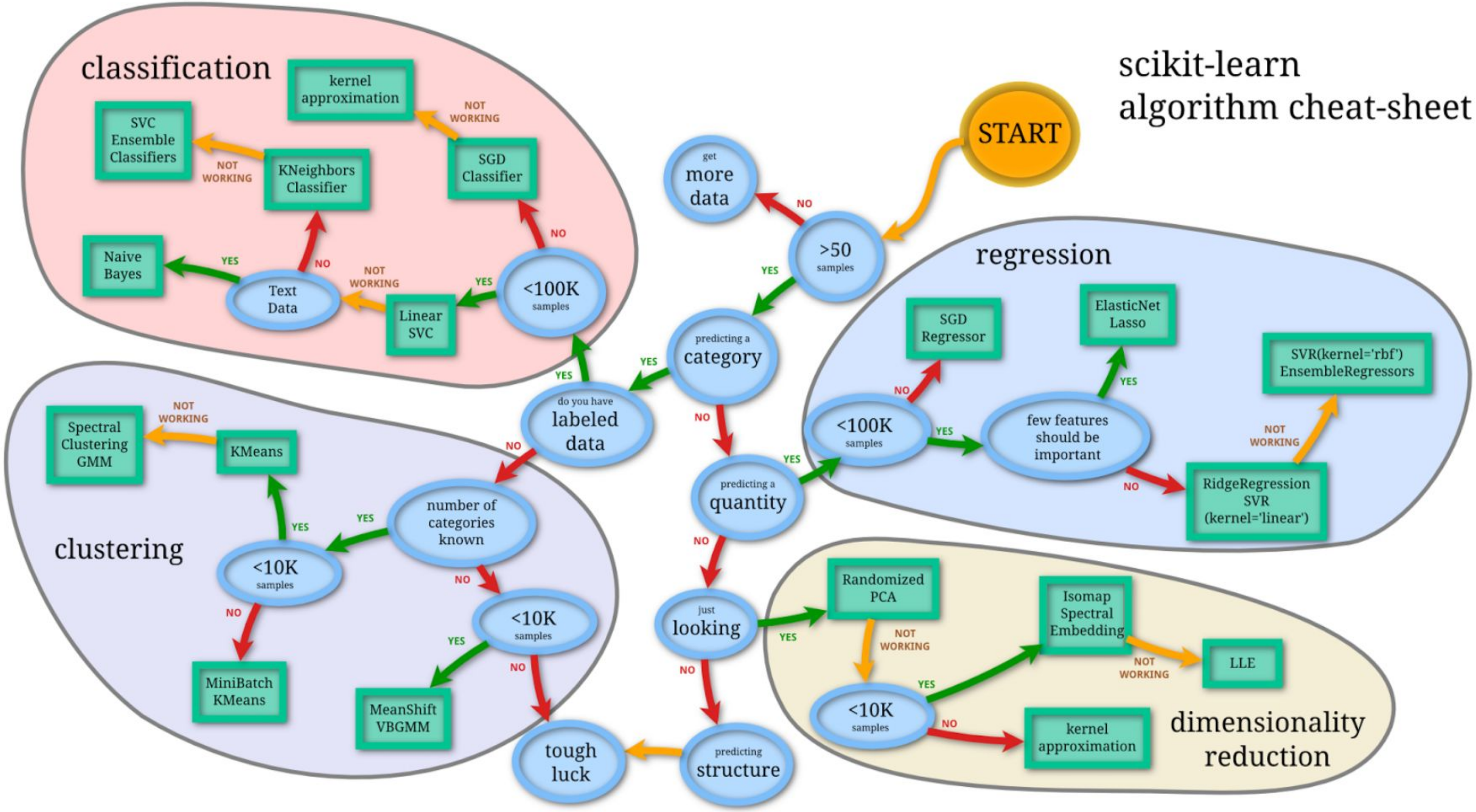


Better than state of the art in less than 30sec!

Why does ML development take a lot of time?



Design-Decisions? That's not how it works!



source:
https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

Hyperparameters are everywhere

SGD

CLASS torch.optim.SGD(*params, lr=0.001, momentum=0, dampening=0, weight_decay=0, nesterov=False, *, maximize=False, foreach=None, differentiable=False, fused=None*) [SOURCE]

Parameters

- **params** (*iterable*) – iterable of parameters to optimize or dicts defining parameter groups
- **lr** (*float, optional*) – learning rate (default: 1e-3)
- **momentum** (*float, optional*) – momentum factor (default: 0)
- **weight_decay** (*float, optional*) – weight decay (L2 penalty) (default: 0)
- **dampening** (*float, optional*) – dampening for momentum (default: 0)
- **nesterov** (*bool, optional*) – enables Nesterov momentum (default: False)

Hyperparameters are everywhere

stable

Search docs

- Installation Guide
- Building From Source
- Get Started with XGBoost
- XGBoost Tutorials
- Frequently Asked Questions
- XGBoost User Forum
- GPU Support
- XGBoost Parameters
- Prediction
- Tree Methods

Python Package

Scikit-Learn API

Scikit-Learn Wrapper interface for XGBoost.

class `xgboost.XGBRegressor`(**, objective='reg:squarederror', **kwargs*)

Bases: `XGBModel`, `RegressorMixin`

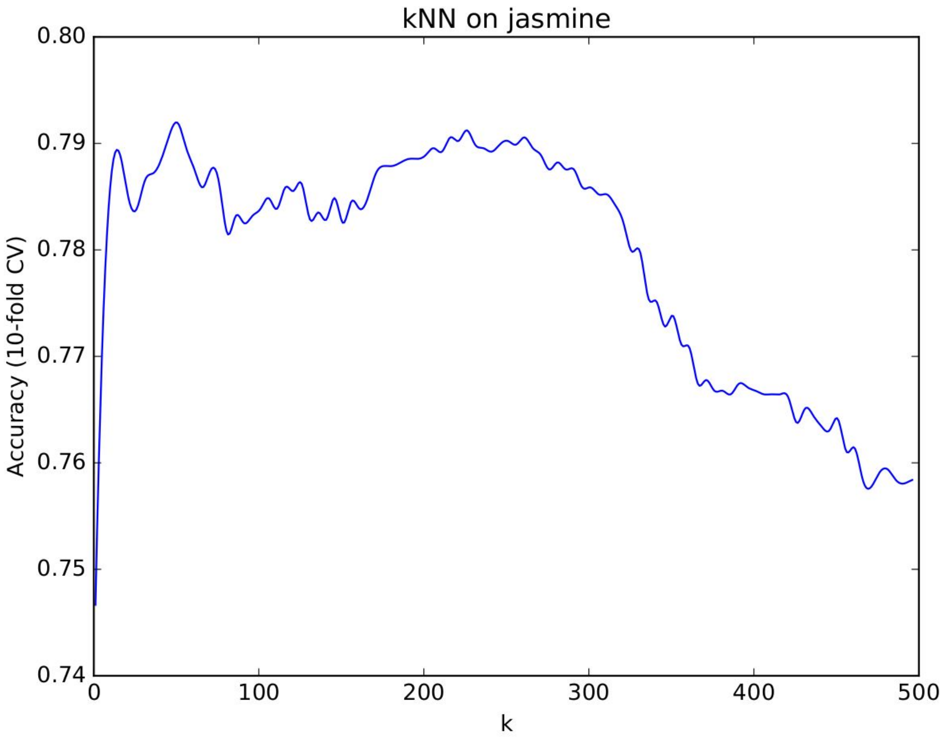
Implementation of the scikit-learn API for XGBoost regression. See Using the Scikit-Learn Estimator Interface for more information.

Parameters:

- **n_estimators** (*Optional[int]*) – Number of gradient boosted trees. Equivalent to number of boosting rounds.
- **max_depth** (*Optional[int]*) – Maximum tree depth for base learners.
- **max_leaves** (*Optional[int]*) – Maximum number of leaves; 0 indicates no limit.
- **max_bin** (*Optional[int]*) – If using histogram-based algorithm, maximum number of bins per feature
- **grow_policy** (*Optional[str]*) – Tree growing policy.
  - depthwise: Favors splitting at nodes closest to the node,
  - lossguide: Favors splitting at nodes with highest loss change.
- **learning_rate** (*Optional[float]*) – Boosting learning rate (xgb's "eta")

Simplest Example Ever: kNN

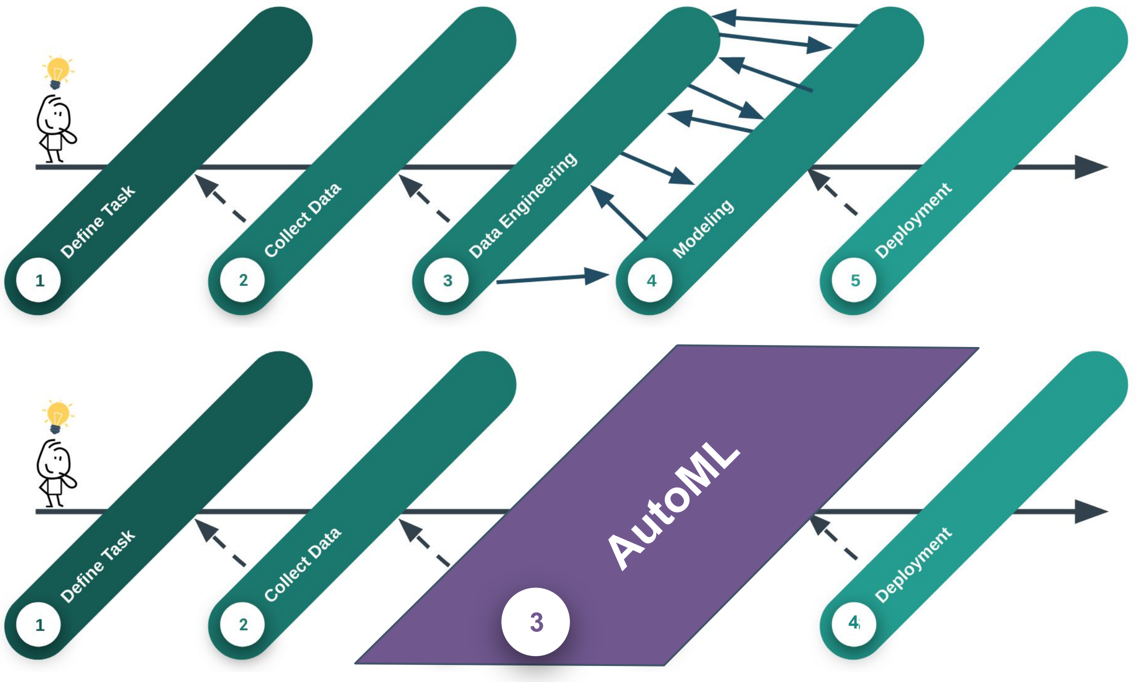
- *k*-nearest neighbors (kNN) is one of the **simplest ML algorithms**
- Size of neighbourhood (*k*) **is very important for its performance**
- The performance function depending on *k* is **quite complex** (not at all convex)
- BUT: We cannot get these curves in practice!



From ML Alchemy to Science

"You can teach an old dog new tricks" [Ruffinelli et al. 2020]
→ Hyperparameter optimization might not be the only required solution, but without it, it will also be hard.

ML vs AutoML



1
Define Task
2
Collect Data
3
Data Engineering
4
Modeling
5
Deployment
1
Define Task
2
Collect Data
3
AutoML
4
Deployment

Topics of AutoML

- **model selection** (e.g., Neural Architecture Search, ensembling)
- **configuration/tuning** (e.g., hyperparameter optimization via evolutionary algorithms, Bayesian optimization)
- **AutoML methodologies** (e.g., reinforcement learning, meta-learning, in-context learning, warmstarting, portfolios, multi-objective optimization, constrained optimization)
- **pipeline automation** (e.g., automated data wrangling, feature engineering, pipeline synthesis, and configuration)
- **automated procedures for diverse data** (e.g., tabular, relational, multimodal, etc.)
- **ensuring quality of results in AutoML** (e.g., fairness, interpretability, trustworthiness, sustainability, robustness, reproducibility)
- supporting **analysis and insight** from automated systems

Advantages

AutoML enables

- 🚀 More **efficient** research and development of ML applications
  → AutoML has been shown to outperform humans on subproblems
- 🧮 More **systematic** research and development of ML applications
  → no (human) bias or unsystematic evaluation
- 📋 More **reproducible** research
  → since it is systematic!
- 🦾 **Broader use** of ML methods
  → less required ML expert knowledge
  → not only limited to computer scientists

Challenges

But, it is not that easy, because

- 🔁 Each dataset potentially requires **different optimal ML-designs**
  - → Design decisions have to be made for each dataset again
- ⌛ Training of a single ML model can be **quite expensive**
  - → We can not try many configurations
- ❓ Mathematical **relation** between design and performance is (often) **unknown**
  - → Gradient-based optimization not easily possible
- 🎛️ 🎚️ Optimization in **highly complex spaces**
  - → including categorical, continuous and conditional dependencies

AutoML Packages

A brief History of AutoML Packages

- **2004**: Ensemble Selection from a set of models
- **2012**: AutoWeka: Efficient in a hierarchy of models and hyperparameters
- **2015**: Auto-Sklearn: Warmstarting of AutoML and ensembling
- **2016**: TPOT: Tree-based Pipeline Optimization for AutoML
- **2019**: AutoKeras: AutoML for Deep Learning
- **2020**: AutoGluon: Stacked Ensemble for diverse Models
- **2021**: AutoPyTorch: Multi-Fidelity for AutoDL powered by SMAC3
- **2022**: TabPFN: In-Context Learning of Tabular ML

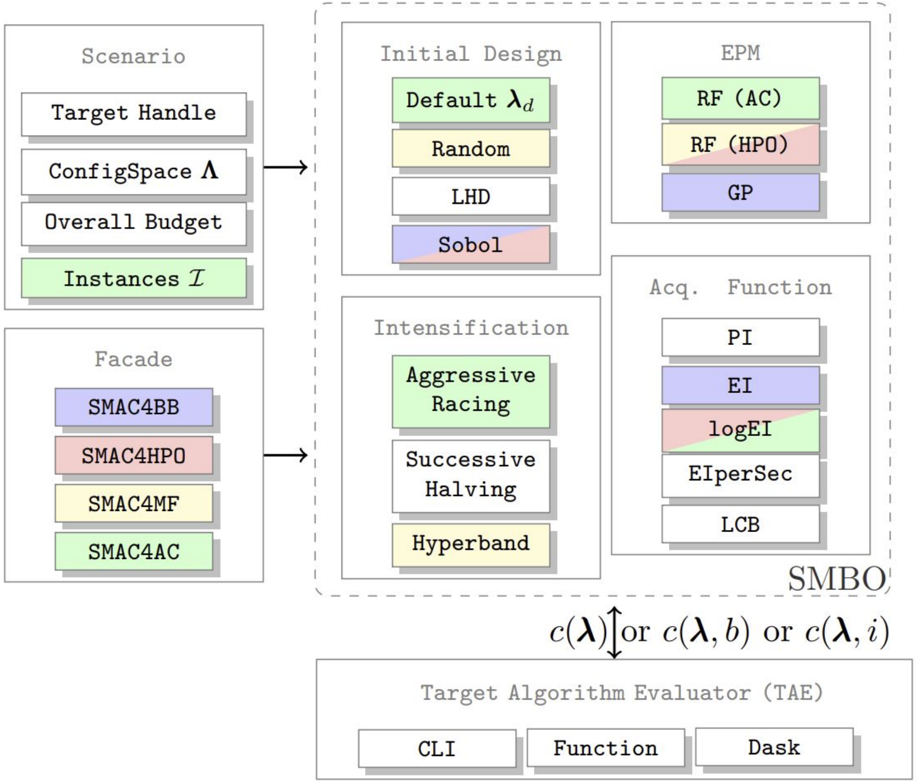
Credits to Matthias Feurer for this list automl / SMAC3 Public

SMAC

Sponsor · Notifications · Fork 220 · Star 1.1k

Code · Issues 95 · Pull requests 6 · Actions · Projects 1 · Security · Insights

- **Use case**: there exits a code base for DS/ML that needs further improvements
- Covers all kind of hyperparameter optimization use cases
- State-of-the-art techniques and performance
    - Bayesian Optimization
    - Multi-fidelity optimization
    - Multi-objective optimization
- Highly configurable and modular
- Parallelizable



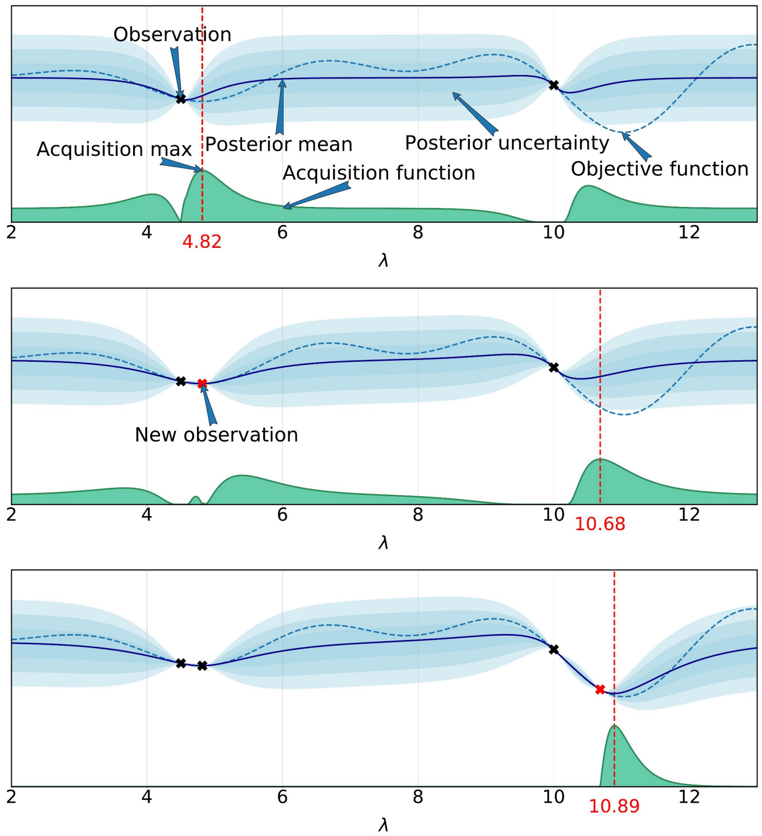
Bayesian Optimization in a Nutshell

General approach:

1. Fit a probabilistic model to predict the performance of unknown Hyperparametrer configurations
2. Trade-off exploration and exploitation for choosing the next configuration to be evaluated

Comments:

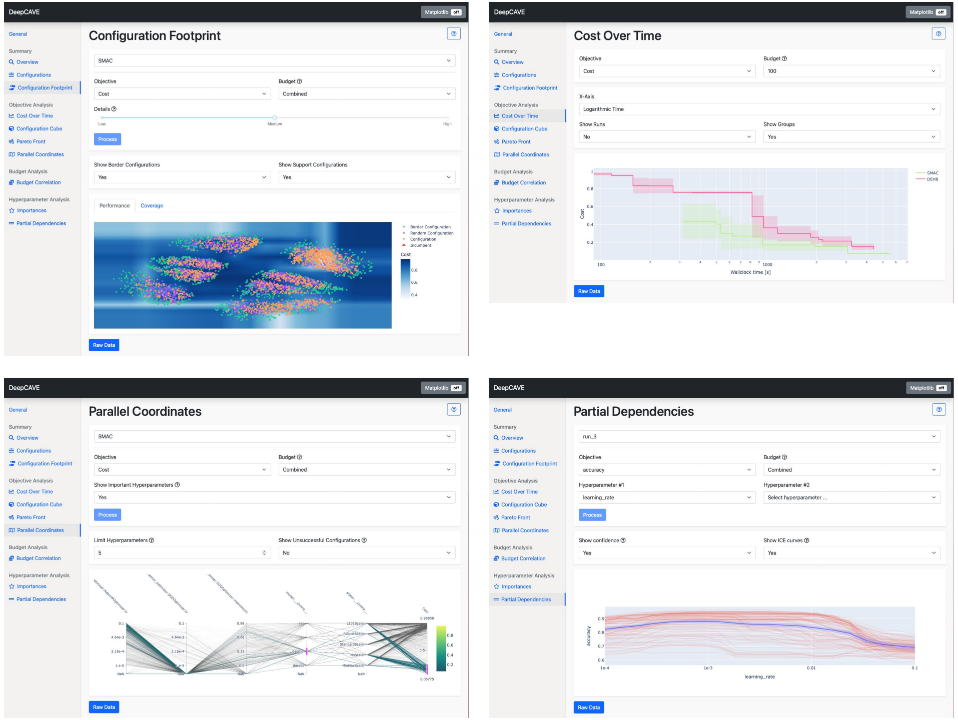
- very efficient in number of evaluations; **<u>much more efficient than grid search and random search</u>**
    - random search is competitive if you have thousands of CPUs/GPUs
- There are convergence results for Bayesian Optimization

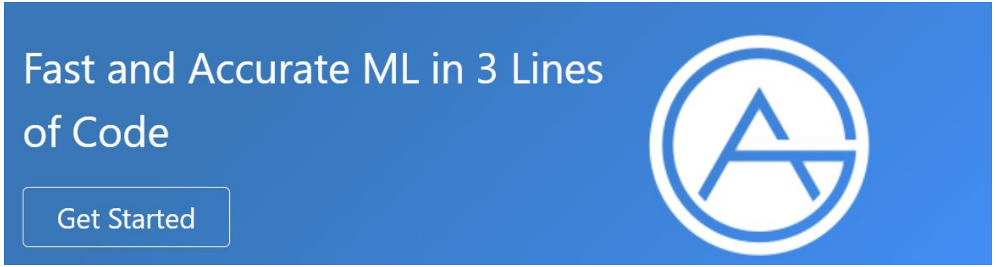


DeepCAVE

DeepCAVE [Sass et al. 2022]

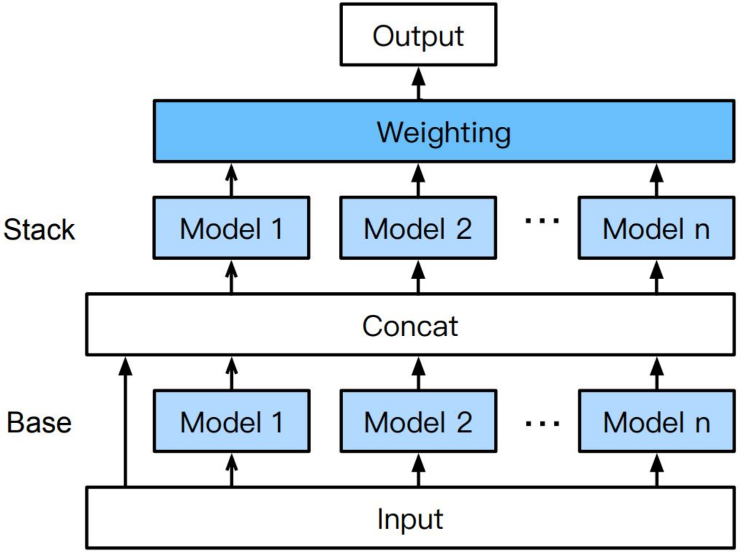
- **Interactive Dashboard** to analyze optimization runs/processes.
- **Exploration of multiple areas** like performance, hyperparameter and budget analysis.
- **Modularized plugin** structure with access to selected runs/groups to provide maximal flexibility.
- **Asynchronous execution** of expensive plugins and caching of their results.
- **API mode** gives you full access to the code





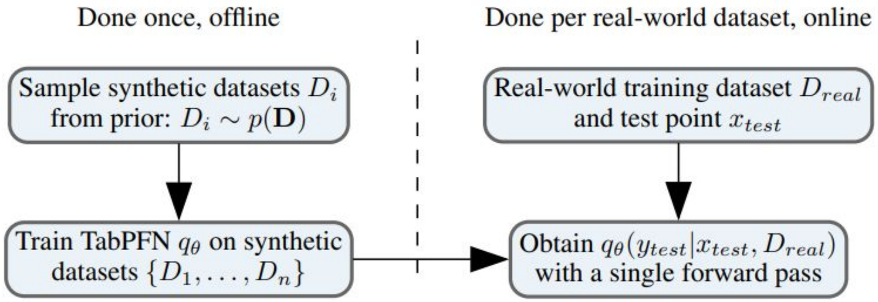
AutoGluon [Erickson et al. 2020]

- **Use case**: Push-Button to obtain a very good model
- Multi-level Stacking instead of Hyperparameter Optimization
  - under restricted time budgets, time is better invested in thoroughly evaluating good defaults settings
  - Inference time can be large
- Hand-defined list of hyperparameter configurations
- Default machine learning model at AWS



TabPFN [Hollmann et al. 2022]

- **Use case**: Training is a bottleneck and training data has a reasonable size
- Based on meta-learned transformers
  - offline training is done of synthetic datasets
- **No training for new data required!**
  - New training data are part of the context input
- No hyperparameter optimization needed
- Very efficient because of missing training
  - but can be limited in data complexity
  - inference time depends on training data (i.e. context) size



(a) Prior-fitting and inference

Find Us

AutoML.org

automl-org

automl

@AutoML_org

LUH|AI

luh-ai

LUH-AI

@luh-ai

Funded by:

European Research Council
Established by the European Commission

DFG
Deutsche Forschungsgemeinschaft

L3S

DAAD
Deutscher Akademischer Austauschdienst
German Academic Exchange Service

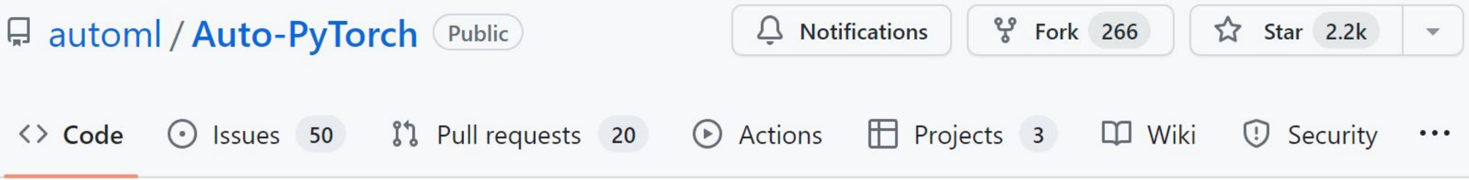
Federal Ministry
of Education
and Research

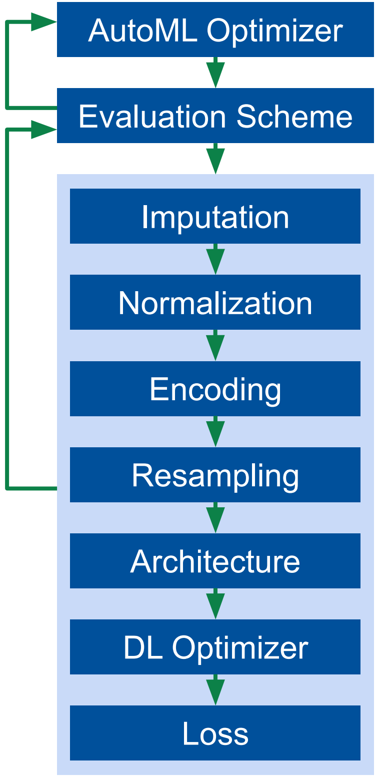
Federal Ministry
for Economic Affairs
and Energy

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

Niedersachsen

Appendix





1. Automatic deep learning covering the entire DL pipeline
2. Joint hyperparameter and neural architecture search

➜ Auto-PyTorch Tabular [Zimmer et al. IEEE TPAMI'21]

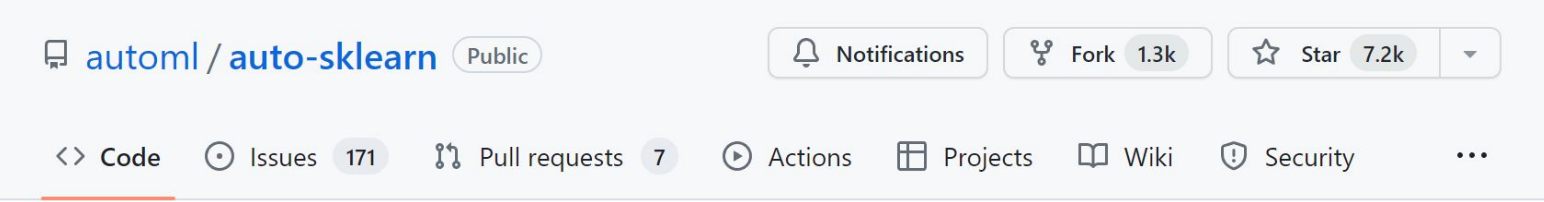
➜ State-of-the-art on tabular data with regularization cocktails [Kadra et al. NeurIPS'21]

➜ Auto-PyTorch for Time Series Forecasting [Deng et al. ECML'22]

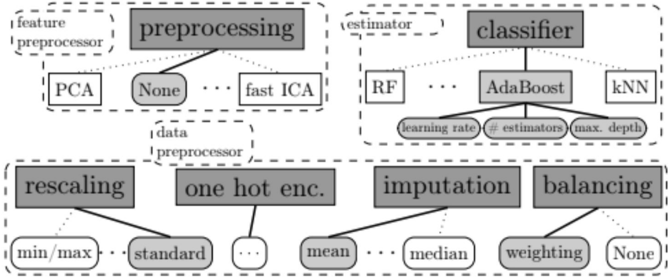
```
# initialise Auto-PyTorch api
api = TabularClassificationTask()

# Search for an ensemble of machine learning algorithms
api.search(
    X_train=X_train,
    y_train=y_train,
    X_test=X_test,
    y_test=y_test,
    optimize_metric='accuracy',
    total_walltime_limit=300,
    func_eval_time_limit_secs=50
)

# Calculate test accuracy
y_pred = api.predict(X_test)
```



Takes care of finding well-performing ML-pipeline



Easy-to-use

```python
import autosklearn.classification as cls
automl = cls.AutoSklearnClassifier()
automl.fit(X_train, y_train)
y_hat = automl.predict(X_test)
```